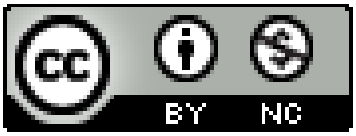
Proyecto final DAW: Blog con Angular y API RESTFUL en PHP

Tecnologías de programación usadas:

Autor: Juan Molina Gómez
Web del autor : https://www.pcmant.com/
Curso: DAW 2
Fecha: 14/06/2022
Lugar: IES Arcipreste de Hita – Azuqueca de Henares (Guadalajara)



Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial 4.0 Internacional.

Índice

1. Definición de proyecto....4
1.1 Tipos de Empresas / Sectores Productivos....4
1.2 Descripción del problema....4
1.3 Objetivos. Especificación de Requisitos....4
1.4 Herramientas de diseño....7
1.5 Estudio de viabilidad....8
1.5.1 API REST....9
1.5.1.1 ¿Qué es una API RESTFUL?....10
1.5.1.2 Características de una API REST....11
1.5.1.3 Formato JSON....12
1.5.1.4 STATUS....12
1.5.1.5 Tipos de autorizaciones en una API REST....12
1.5.1.6 Ventajas de servicios de terceros para autenticarse y obtener token....13
1.5.2 JSON Web Token (JWT)....13
1.5.3 MySQL....14
1.5.4 PHP con MySQL....15
1.5.5 Angular....16
2. Planificación del Proyecto:....21
2.1 Secuenciación de las fases del proyecto....21
2.2 Planificación de recursos y tiempos....22
3. Análisis del Proyecto:....23
3.1 Casos de uso....23
3.2 Modelo de datos....23
3.2.1 Esquema Entidad / Relación....25
3.2.2 Diagrama UML de la base de datos....25
3.2.3 Modelo relacional y normalización....25
3.2.4 Diccionario de datos....26
4. Diseño del Proyecto:....29
4.1 Diseño del modelo Vista/Controlador....29
4.2 Diseño físico de la BD y estructura de los ficheros....30
4.2.1 Diseño físico de la BD....30
4.2.2 Estructura de los ficheros....30
4.2.2.1 anglularBlogProyectoDaw....32
4.2.2.2 maquetacionProyectoDaw....36
4.2.2.3 PhpApiRestBlogProyectoDaw....37
4.3 Diseño de la estructura de clases y librerías (diagrama de clases)....39
4.3.1 Diagrama UML API RESTFUL con PHP....39
4.3.2 Diagrama UML de la parte desarrollada con Angular....40
4.3.2.1 Diagrama UML services y models aplicados al proyecto de Angular..40
4.3.2.2 Diagrama UML componentes del proyecto con Angular....41

4.4 Diseño de procedimientos especiales....42
4.5 Diseño de la interfaz gráfica....47
4.5.1 Sketch....47
4.5.2 Wireframe....48
4.5.3 Mockup....49
5. Desarrollo y realización del Proyecto:....55
5.1 Indicadores de calidad....55
5.2 Elaboración de una batería de pruebas para detectar errores....55
5.3 Evaluación y solución de incidencias....59
5.4 Evaluación y seguimiento de las actividades....59
6. Implantación del Proyecto:....60
6.1 Plan de implantación....60
6.2 Manual de instalación....60
6.2.1 Instalación en local para pruebas o desarrollo....60
6.2.1.1 Instalación de las dependencias para el entorno local desde Windows 60
6.2.1.2 Instalación de Angular CLI para el entorno local....77
6.2.1.3 Instalación de Postman y importación de una colección de datos....80
6.2.1.4 Despliegue de la aplicación en local....81
6.2.2 Configuraciones y ajustes en el proyecto muy importantes a tener en cuenta 82
6.2.3 Instalación en un servidor web para producción....88
6.2.3.1 Como poner en producción la base de datos....88
6.2.3.2 Como poner en producción la API RESTFUL en un servidor Apache con PHP....88
6.2.3.3 Como poner en producción Angular para un servidor Apache....88
6.3 Manual de usuario....90
6.3.1 Como Iniciar sesión (Imprescindible para administrar el blog)....90
6.3.2 Como crear un artículo nuevo (Inicio de sesión requerido)....91
6.3.3 Como Editar/Actualizar un artículo (Inicio de sesión requerido)....93
6.3.4 Como eliminar un artículo (Inicio de sesión requerido)....96
6.3.5 Como cerrar la sesión....97
6.3.6 Uso de Postman....97
7. Conclusiones Finales:....99
7.1 Evaluación del proyecto: objetivos conseguidos y no alcanzados....99
7.2 Propuestas de mejora....100
8. Biografía....101

1. Definición de proyecto

Está aplicación web se trata de un sistema de blog hecho con Angular como frontend y en el backend una API RESTFUL dinámica desarrollada en PHP nativo utilizando una base de datos MySQL.

1.1 Tipos de Empresas / Sectores Productivos.

Este producto de software se adapta a cualquier tipo de empresa o particular que precise de una web de presentación con un blog integrado, también es aplicable a un portal de noticias.

A su vez también tiene como objetivo otros desarrolladores debido a que el backend de la aplicación web tiene una API RESTFUL dinámico, ya que **las APIS son aplicaciones creadas para programadores**, pues carecen de interfaz frontend y se utilizan para obtener información en formato XML o JSON principalmente (como en este caso) e interactuar con la API.

1.2 Descripción del problema.

Esta aplicación web busca cubrir la gran demanda que tiene las aplicaciones de tipo SPA, las cuales ofrecen una experiencia de usuario muy buena y superior a las webs típicas monolíticas, las cuales al cambiar de ruta recarga su web, cosa que no ocurre con las **SPA las cuales nunca recargan la página, son reactivas, instantáneas** y nunca recargan la página por más que cambies de ruta o interactúes con la página, **ofreciendo una mayor velocidad de carga y una mejor experiencia de usuario.**

1.3 Objetivos. Especificación de Requisitos.

Esta aplicación tiene el objetivo de ser una web SPA con un sistema de Blog utilizando MySQL como base de datos, API RESTFUL dinámica con PHP y consumiendo dicha API y mostrando el frontend con Angular.

En cambio desde el lado del usuario tiene dos objetivos principales:

- Desde el lado **usuario invitado** (El cual es el usuario promedio de está aplicación): navegar por la aplicación web y visualizar los artículos del blog.

- Desde el lado del **usuario registrado**: Administrar los artículos del blog, creando, editando y eliminando los mismos.

Especificaciones técnicas a nivel de software del entorno de desarrollo necesarios para poder ejecutar la aplicación web:

- MySQL 5.7.31
- Apache 2.4.48
- PHP 7.4.23
- Node.js v16.14.2
- Angular CLI 13.3.3

Especificaciones técnicas mínimas a nivel de hardware y software necesarios para poder funcionar correctamente cada una de las partes de la aplicación

- **Requisitos MySQL 5.7.31**
  - CPU con 2 núcleos
  - 2 GB RAM
  - Subsistema de E/S de disco aplicable a una base de datos de escritura intensiva
  - Sistemas operativos: Linux/Unix, Microsoft Windows y MacOS.
- **Requisitos Apache 2.4.48**
  - **En Windows**:
    - La plataforma principal de Windows para ejecutar Apache 2.4 es Windows 2000 o posterior. Obtenga e instale siempre el Service Pack actual para evitar errores en el sistema operativo.

        - Las versiones del servidor Apache HTTP posteriores a la 2.2 no se ejecutarán en ningún sistema operativo anterior a Windows 2000.

    - En sistemas **tipo Unix** (Como Linux o MacOs):

        - APR y APR-Util

        - Librería Compatible de expresiones regulares de Perl (PCRE)

        - Espacio en disco mínimo de 50 MB.

        - Sistema de compilación ANSI-C

        - Ajuste exacto del reloj del sistema

- **Requisitos PHP 7.4.23**

    - PHP requiere al menos Windows 2008/Vista. Ya sea 32 bits o 64 bits (AKA X86 o X64. PHP no se ejecuta en Windows RT/WOA/ARM). A partir de PHP 7.2.0 Windows 2008 y Vista ya no son soportados.

    - PHP requiere Visual C runtime (CRT). Muchas aplicaciones lo requieren por lo que ya esté instalado.

    - El Microsoft Visual C++ Redistributable para Visual Studio 2019 es adecuado para todas las versiones de PHP.

    - Debes descargar la biblioteca x86 CRT para compilar PHP para arquitecturas x86 y x64 CRT para compilar de PHP x64.

    - Si CRT ya está instalado, el instalador te lo dirá y no cambiará nada.

    - El instalador de CRT admite los modificadores de línea de comandos /quiet y /norestart, por lo que puede ejecutar un script.

- **Node.js v16.14.2**

    - 2GB de RAM

- Al menos 4 vCPU
- Al menos 25 GB de almacenamiento

- **Angular CLI 13.3.3**
  - Node.js (14.15.0 o 16.10.0)
  - Gestor de paquetes npm. (Imprescindible para poder instalar Angular y dependencias, suele venir acompañado con la instalación de Node.js, que es el que se encarga de que se ejecute Angular).

Por último en el aspecto técnico si eres desarrollador **es recomendable la utilización de una aplicación de escritorio llamada Postman el cual nos servirá como cliente HTTP para poder probar la API haciendo peticiones a ella**, con el proyecto viene adjunto un fichero preparado con una batería de peticiones HTTP de prueba y la cual documenta sobre el uso de la API, una documentación muy importante para todo desarrollador que se precie para que sus aplicaciones puedan interactuar con la API.

1.4 Herramientas de diseño.

En este apartado se enumeran las herramientas necesarias para el diseño y la correcta composición de la aplicación web:

- **Entorno frontend** (el cliente):
  - Angular 13 y las siguientes librerías:
    - Bootstrap 5.1.3 (frameworks de CSS con librería de estilos)
    - Bootstrap-icons 1.8.2 (Librería de iconos)
    - ngx-moment v6.0.2 (formatea fechas)
    - angular-jwt 5.0.2 (Para comprobar tokens bajo el estándar jwt, en mi caso lo uso para saber si ha caducado)

- Entorno **backend** (el servidor):
  - MySQL 5.7.37
  - Apache 2.4.48
  - PHP 7.4.23, acompañado de las siguientes librerías:
    - firebase/php-jwt (Para crear tokens bajo el estándar jwt).
  - **Postman** (Aplicación de escritorio para testear y probar la API, esto es opcional, pero muy importante tenerla para hacer pruebas con la API y hacer cosas aun no implementadas en la aplicación web desarrollada con Angular).

1.5 Estudio de viabilidad

Debido a la naturaleza del proyecto **no ha habido gasto económico** salvo gasto de tiempo y mano de obra por parte del desarrollador del proyecto, es totalmente manteniente, sin conocimientos en desarrollo es posible mantener el blog a excepción de que se quieran editar las páginas estáticas que no sean artículos, las rutas lo cual conllevaría tener o mejorar la funcionalidad como mínimo conocimientos básicos sobre Angular y JavaScript.

Respecto al factor tiempo, la idea original ha sido adaptada al tiempo limitado para que sea viable de hacer.

Respecto a temas legales y derechos de autor, se ha utilizado tecnologías y herramientas de software libre (en su mayoría) y código abierto para la elaboración de dicho proyecto, herramientas del sistema y software freeware (aunque no sea libre significa que es de uso gratis), además el software desarrollado en este proyecto ha sido licenciado con una licencia **GNU General Public License v3.0**, para conocer los detalles podrás verlo en el siguiente enlace https://choosealicense.com/licenses/gpl-3.0/.

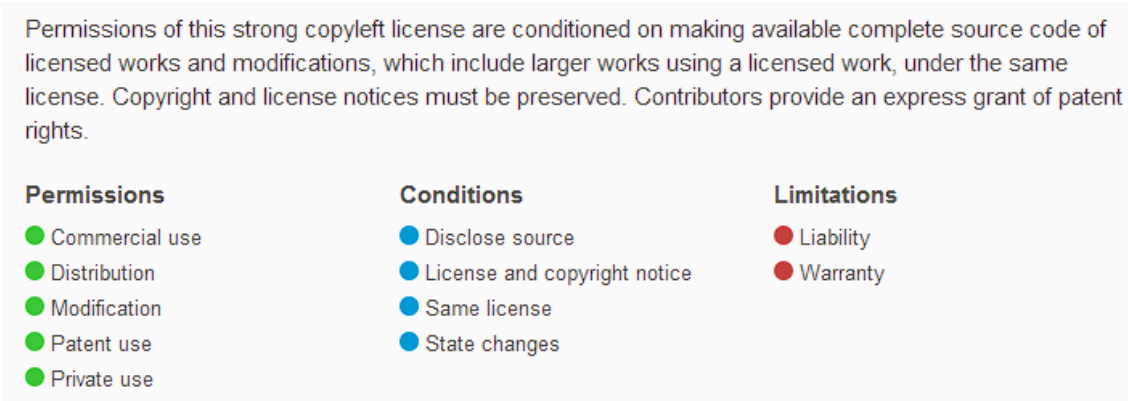


Imagen 1.5.1: Condiciones generales de la licencia GNU General Public License v3.0

A continuación respecto a la **viabilidad tecnológica** se aportará información y se analizará el porqué de las tecnologías elegidas con el fin de justificar su uso y informar al lector.

1.5.1 API REST

Hoy en día la mayoría de empresas que tienen servicios en la nube que utilizan **APIs REST**, esto se debe a que es un estándar lógico y eficiente para la creación de servicios web.

- **API**: Aplication Programming Interface (Interfaz de programación de aplicaciones).
- **REST**: Aplication Programming State Transfer (Transferencia de estado representacional)
- **API REST**: Interfaz de aplicaciones para transferir datos.

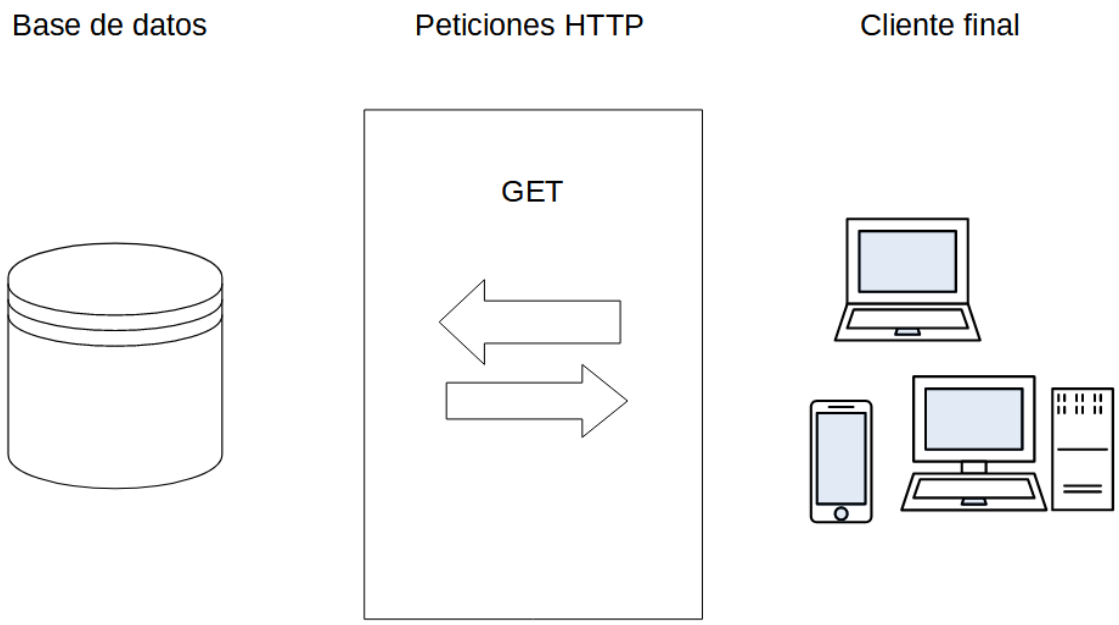


Imagen 1.5.1.1: Diagrama descriptivo del funcionamiento de una API REST

1.5.1.1 ¿Qué es una API RESTFUL?

Básicamente es una aplicación API de tipo REST que es capaz no unicamente de hacer peticiones GET, si no de varios tipos de peticiones HTTP como: POST, PUT y DELETE.

Una API RESTFUL se convierte a aquellas empresas que necesiten que su base de datos sea constantemente cambiante (insertando datos, creando datos, actualizando datos), eso ya dependiendo de la empresa o la necesidad se construye.

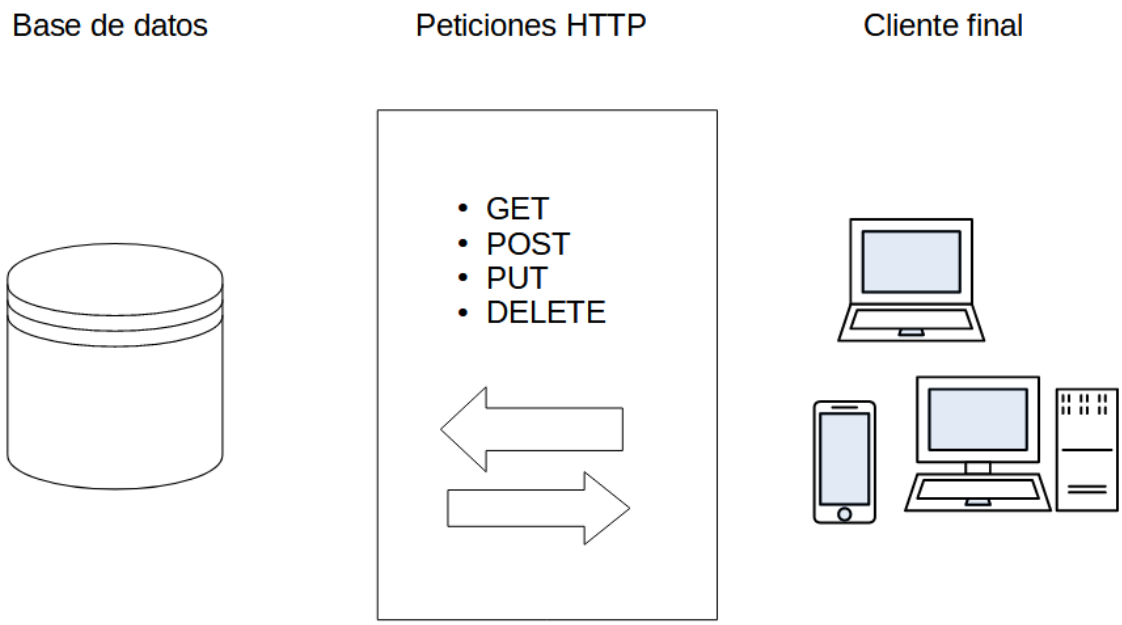


Imagen 1.5.1.1.1: Diagrama descriptivo de como funciona una API RESTFUL

1.5.1.2 Características de una API REST

- Son desarrolladas por programadores para programadores.
- Públicas.
- Privadas.
- Gratuitas.
- De pago.
- Algunas APIs están protegidas con token de seguridad, podría ser útil para proteger la base de datos.
- Lenguajes web para construir una API REST: PHP y sus frameworks, JavaScript lado de servidor con Node.js.
- **Es indispensable crear una documentación para su uso.**

1.5.1.3 Formato JSON

Para poder hacer uso de las APIs REST es muy importante conocer lo que es el formato JSON.

JSON: JavaScript Object Notation (Notación de objeto de JavaScript).

Es un estándar basado en texto plano para el intercambio de información, por lo que se usa en muchos sistemas que requieren mostrar o enviar información para ser interceptada por otros sistemas.

El formato JSON están construido por array y objetos, te permite escribir datos sin limites.

A las aplicaciones el JSON le llega como una cadena de texto, por lo cual hay que parsear.

1.5.1.4 STATUS

Los estados más conocidos:

- **400 Not Found**: No encontrado, indica que el servidor no puede encontrar el recurso solicitado.
- **500 Internal Server Error**: Error en el servidor o situación que no puede ser manejada por el mismo.
- **200 OK**: Indica que la solicitud ha tenido éxito, una respuesta 200 es almacenable de forma predeterminada.

1.5.1.5 Tipos de autorizaciones en una API REST

En este proyecto debido a su sencillez de uso la parte privada de la API utiliza como modo de autenticación el tipo API KEY, que básicamente consiste con un token enviado en las cabeceras de las peticiones HTTP.

Hay varios tipos de autenticación:

- **OAUTH**: “Open Authorization” es un estándar abierto que permite flujos simples de autorización para sitios web o aplicaciones informáticas. Es una

simple combinación de caracteres que convierte en una poderosa contraseña. (la más sencilla, consiste pasar por cabecera la key Authorization con el token como value).

- **OAUTH 2**: Open Authorization 2 es un flujo de autorización específico encapsulado en un token portador (bearer token) y se genera con la combinación de unas credenciales que le suministra la empresa al cliente llamadas CLIENT ID y SECRET KEY. (da un poco más de seguridad, da seguridad a la base de datos, es una combinación mucho más extensa entre client id y clave secreta).

  En algunas ocasiones utilizan el encriptado de tipo md5() o base64_encode() para combinar las credenciales del cliente y generar el Bearer Token.

- **API KEY**: Un API Key es un identificador que sirve como el medio de autenticación de un usuario para el uso de los servicios proporcionados de una API REST. Este valor es único y el cliente debe solicitarlo al proveedor de la API.

La autenticación puede ser por:

- **AUTENTICACIÓN POR CABECERA** (Headers): Es cuando el token de autorización debe ser pasado de forma oculta por cabeceras HTTP.

- **AUTENTICACIÓN POR URL**: Es cuando el token de autorización debe ser pasador a través de la URL.

1.5.1.6 Ventajas de servicios de terceros para autenticarse y obtener token

- Correo electrónico válido.

- No tenemos manipulación de contraseña.

- Nos devuelven datos como por ejemplo: Nombre completo, foto...

1.5.2 JSON Web Token (JWT)

Es un algoritmo encriptado basado en JSON que permite el intercambio y la verificación de datos entre aplicaciones o servicios de forma segura.

Sirve para que el cliente pueda hacer solicitudes a la API, solo si tiene un token válido solo esa persona podrá hacer peticiones.

Los tokens constan de tres partes para que sea único e irrepetible:

- **Cabecera**. Este contiene el algoritmo de cifrado.
- **Datos**. El cuerpo o la carga útil de los datos, aquí solo contiene datos útiles del usuario, contiene datos identificativos únicos del usuario y el iat (fecha de expiración del token.
- **Verificación de la firma**.

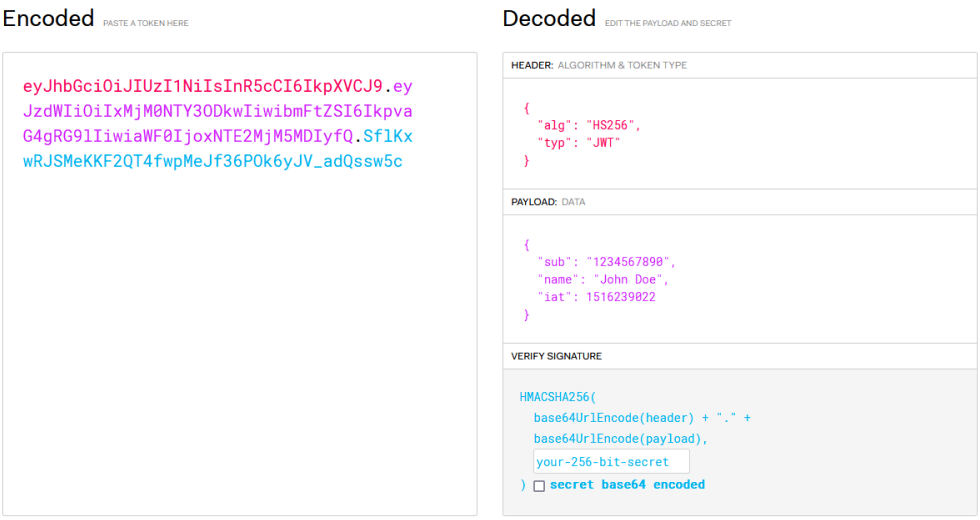


Imagen 1.5.2.1: Estructura de un token jwt, utilizando la herramienta de la web https://jwt.io/

Cada una de estás partes viene encriptadas para que sea único e irrepetible y este completamente seguro para poder ser usado entre aplicaciones y servicios.

1.5.3 MySQL

Razones por las cuales elegir MySQL como gestor de base de datos:

- Es rápido en la escritura de datos y mayor, se nota más la diferencia de las NoSQL cuando se trata de base de datos grandes.

- Tiene el inconveniente de ser más enrevesados que su contraparte NoSQL en la recuperación de datos ya que es posible que se tenga que buscar los datos entre varias tablas, puede llegar a ser un poco más lento, pero solo se llega a notar en grandes escalas, pero en este caso que es una base de datos relativamente pequeña ni si quiera se va a notar.

- Son robustas.

- Es multiplataforma.

- Es fácil de encontrar ayuda.

- Es fácil de aprender.

- MySQL es una base de datos ampliamente probada por distintos usuarios y empresas.

- Es una base de datos de código abierto, no tiene coste monetario.

- Utiliza varias capas de seguridad. Contraseñas encriptadas, derechos de acceso y privilegios para los usuarios.

- Necesita pocos recursos de CPU o RAM y tiene una baja fuga de memoria.

1.5.4 PHP con MySQL

Razones para usar PHP con MySQL:

- Más del 50% de las webs están hechas con PHP y MySQL.

- Ambos son gratuitos.

- La curva de aprendizaje es muy baja.

- PHP tiene muy buena interacción con HTML (En este caso no se va a usar en PHP debido a que monto la API unicamente devuelve información en formato JSON, pero no deja de ser un punto a favor).

- Tiene una comunidad muy amplia donde resolver dudas, tanto oficial como extraoficialmente.

- Los entornos de desarrollo de PHP y MySQL son fáciles de utilizar y configurar.

- Son lenguajes fiables, eficientes y fáciles de usar.

- PHP tiene un fácil acceso a software de terceros, es decir, si tenemos que trabajar con APIs o con software de otros clientes que está realizando en PHP, se integra muy fácilmente con nuestros desarrollos.

- MySQL es una de las bases de datos más potentes y populares que hay en el mercado.

1.5.5 Angular

Es uno de los frameworks más importantes de JavaScript que existen actualmente, mantenido por Google y nos permite hacer webs SPA, dicho framework es orientado a componentes (por eso puede funcionar como SPA) y tiene un tipado fuerte debido a que usa la sintaxis de TypeScript (Es como un JavaScript pero con tipado, este es compilado a un JavaScript).

> "Un **framework** es un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software" (https://www.edix.com/).

Una SPA (Single Page Application) es una página web en la que unicamente tienes una página y la interacción entre las diferentes secciones es completamente reactiva, es decir, que la navegación entre entre entre esas diferentes páginas va por JavaScript, es completamente instantánea, es decir, que la página no recarga. Todo el tratamiento, todo el trabajo con las distintas rutas y las URLs de la página también va por JavaScript, entonces si navegas entre diferentes paginas a nivel de interacción de usuario, pero realmente todo el desarrollo de la web es una single page, una página única que no tiene necesidad de recargar toda la página por cada interacción del usuario con ella.

Por ejemplo si en una web cambias de Inicio a Blog, entonces no recargaría toda la página, unicamente el bloque correspondiente a esa parte recargaría.

Toda la interacción con el backend es asíncrona, es decir, va por por AJAX, por peticiones HTTP asíncronas.

Las aplicaciones web de tipo SPA son mucho más modernas, mucho más usables para el usuario, muchísimo más rápidas que una aplicación web monolítica clásica (aplicaciones en las cuales cargan la pantalla completa por cada iteración, por ejemplo una web hecha en PHP).

Conceptos de Angular:

- **Componentes**:
  - Un componente al final va a controlar un trozo de pantalla o de la vista.
  - Todo lo que se puede ver en pantalla es controlado y gestionado por este tipo de elementos.
  - La lógica de un componente dentro de una clase en Angular es que da soporte a una vista interactuando con ella a través de un API con propiedades y métodos.
  - El componente hace de mediador entre la vista a través de la plantilla y la lógica de la app donde se incluirá el modelo de datos, es decir una especie de controlador.
- **Plantillas**:
  - Las plantillas van a definir la vista de los componentes.
  - Son HTMLs y tienen sintaxis especial de Angular. Trabajando con el data binding y las directivas.
- **Decoradores de metadatos**:
  - Con los decoradores (patrón de diseño) vamos a configurar diatónicamente atributos/metadatos de las clases y componentes.

- Los metadatos van a describir a las clases pero también describen relaciones, por ejemplo si tenemos un componente y una plantilla el metadato se va a encargar de decirle a Angular que ese componente y esa plantilla van juntos, entre otras muchas cosas.

- **Servicios**: Son clases con un objetivo claro, facilita la reutilización, son un tipo de elemento dentro de la arquitectura de Angular y mediante la inyección de dependencias lo podremos usar en otros componentes principales digamos.

- **Providers**: Son servicios que nos proveen de datos o funcionalidades mediante sus métodos. Existen providers/servicios propios de Angular creados por nosotros mismos.

- **Directivas**:

  - Son funcionalidades aplicables al DOM y a los elementos HTML en las plantillas de un componente. Por ejemplo una directiva puede servir para controlar que un div se muestre o no o recorrer un array en la vista (directivas estructurales, estructuras condicionales y de control) o incluso también puede servir para darle un estilo u otro a un elemento HTML o también para interactuar con el modelo de los datos del componente.

  - Básicamente son nuevos atributos para aplicarle a cualquier cosa en nuestra plantilla/vista,

- **Módulo**:

  - Actualmente Angular es modular.

  - Se puede tener un módulo para la aplicación principal y dentro de otro un panel de administración, otro para una tienda, otro para la parte publica de la web… Y así puedo crear diferentes módulos que hagan diferentes funcionalidades, aunque estén en el mismo proyecto son módulos separados que nosotros podemos estar reutilizando en otros proyectos, y de ese modo poder estar mejorando la arquitectura de nuestra aplicación web.

Razones para utilizar Angular:

- **Angular.js** (Este dejo de tener soporte recientemente el cual usaba JavaScript puro) **es diferente a Angular CLI** (Mucho más actual, usa TypeScript y es **mantenido por la empresa Google**.

- Angular es uno de los frameworks de JavaScript que más trabajo da

- Permite crear webs SPA (Single Page Aplication), esto quiere decir que es una aplicación web de una sola página, aunque se pueda navegar por las diferentes rutas de la web y cambie los datos dinámica mente es solo una página.

- Es un proyecto web que toda que toda su interactividad es reactiva, instantánea y asíncrona con el servidor.

- Las SPA es un tipo de aplicación que actualmente se está haciendo mucho, debido que la experiencia para el usuario es muy buena.

- Arquitectura y un orden para los proyectos espectacular, no hay un framework dentro de JavaScript que nos den un orden y una arquitectura que nos puede implantar tan buenas prácticas como es Angular.

- Las aplicaciones siempre van a seguir prácticamente los mismos estándares, tiene unas reglas para programar dentro de él.

- Obligatoriamente hay que usa TypeScript dentro de el, se trata de un 90% de JavaScript normal, pero vitaminado con tipo de datos, mejoras en el propio lenguaje de de programación de JavaScript, mejoras en la programación orientada a objetos, mejoras de compatibilidad con los últimos estándares, etc.

- TypeScript es básicamente un poco de JavaScript con tipado fuerte.

- Esto hace que estemos programando, entendiendo mejor acerca de tipos de datos y haciendo una aplicación más robusta en general.

- El routing, dentro de este framework es una autentica gozada, todo el tema de las rutas o las URLs amigables, todo el tema de la navegación entre las diferentes URLs del proyecto.

- El routing permite ser gestionado de una manera increíble y cada uno de estos segmentos o URLs o rutas están vinculadas a un componente, esto hace que carguen las diferentes páginas de manera más rápida e instantánea que una web molifica tradicional.

- El principal uso de Angular es utilizar el sistema de routing integrado para el propósito mencionado anteriormente.

- Permite hacer un tipo de aplicación SPA muy buscada por los clientes y laboral mente.

- La pantalla nunca recarga, es decir, todo lo que aparece por pantalla lo va a hacer en tiempo real de manera instantánea o de manera asíncrona haciendo peticiones al servidor, por lo cual genera un efecto bastante agradable de velocidad, mayor rapidez que una web monolítica y mejor usabilidad para el usuario e mejor experiencia para el mismo.

- Todo va cargando y apareciendo instantáneamente.

- Es reactivo y tiene el **two-way data binding**, esto significa que tu puedes modificar lo que está en el modelo de datos e instantáneamente va a aparecer ese cambio en la interfaz del usuario en la interfaz de la web, siendo este también uno de los usos principales del framework, son cosas que solo se pueden conseguir con frameworks de este tipo.

- Se puede hacer cambios en el DOM sin manipular el DOM directamente, utilizando su sistema de plantillas y usando el sistema de componentes que tiene el framework, lo cual lo hace mucho más fácil y sencillo de usar por el desarrollador sin necesitad de tener que enredar con queryselectors, documentById… y sin tener que lidiar con la complejidad que supondría en JavaScript nativo.

- Los componentes son re utilizables si se desarrollan con ese propósito e incluso reutilizarlos en otros proyectos.

- Módulos re utilizables, los cuales son una aplicación dentro de un mismo proyecto los cuales se pueden reutilizar para otros proyectos, para ello se tendría que desarrollar con ese propósito. (Por ejemplo un panel de administración que pueda usar en varios proyectos sin tener que volverlo a desarrollar de cero).

- Tiene un lenguaje de plantillas increíble, puedes tener un montón de templates, incluyendo una dentro de otra, pesarle información a esas plantillas, tiene su propio bucles, sus propias condiciones, sus propias directivas, sus propios pipes, es muy rico en sí lo que es el tema de manipulación de plantillas en lo que es el uso del HTML dentro del framework y la limpieza que tiene.

2. Planificación del Proyecto:

2.1 Secuenciación de las fases del proyecto.

- Preparar el entorno de programación.
- Diseño de la base de datos.
- Crear la base de datos.
- Formarme acerca de las APIs REST con PHP y crear la API RESTFUL.
- Crear el diseño y realizar la maquetación web.
- Crear el frontend con Angular.
    - Instalación de dependencias.
    - Crear componentes y añadirle la maquetación a sus respectivas partes.
    - Crear rutas.
    - Crear las páginas que son estáticas y el formulario de contacto.
    - Crear modelos y servicios con sus respectivos CRUD con la API.
    - Hacerle las distintas programaciones y adecuación de rutas y componentes al funcionamiento del blog.
        - Listado de los últimos artículos en Home.
        - Listado de todos los artículos en blog.

- Búsqueda de artículos.
- Página de login y comprobación de sesión.
- Creación de artículos, edición y borrado.

- Crear documentación.

2.2 Planificación de recursos y tiempos

		Nombre	Duracion	Inicio	Terminado
1		Formación y recolección de datos	15 days?	4/04/22 8:00	18/04/22 17:00
2		Base de datos	3 days?	19/04/22 8:00	21/04/22 17:00
3		API RESTFUL con PHP	21 days	21/04/22 8:00	11/05/22 17:00
4		Maquetación Interfaz y diseño	1 day?	11/05/22 8:00	11/05/22 17:00
5		Angular	21 days?	11/05/22 8:00	31/05/22 17:00
6		Memorias	14 days	1/06/22 8:00	14/06/22 17:00

Imagen 2.2.1: Listado de planificación de tareas

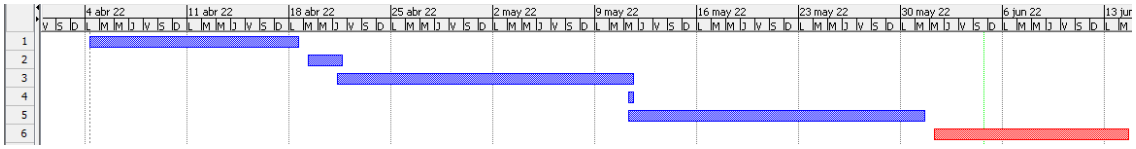


Imagen 2.2.2: Diagrama de la planificación de tareas

Tiempo total: 74 días.

3. Análisis del Proyecto:

3.1 Casos de uso

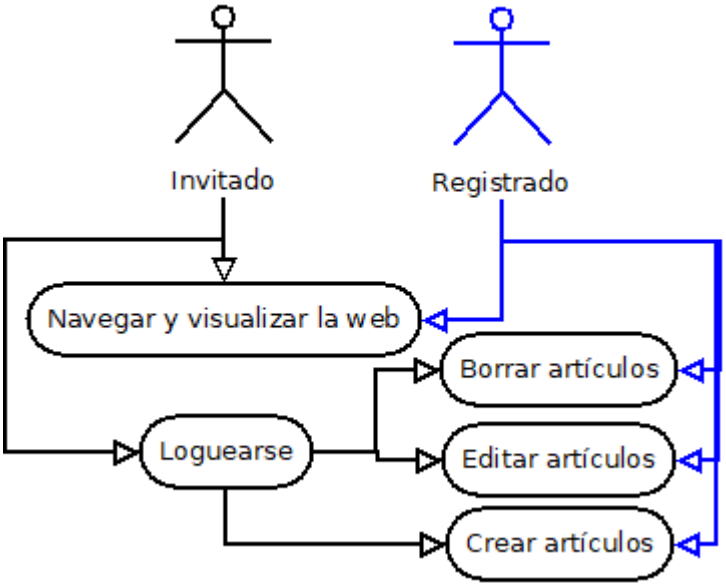


Imagen 3.1.1: Diagrama UML casos de uso desde el punto de vista de la aplicación web hecha con Angular

3.2 Modelo de datos

La API construida en este proyecto es **dinámica** y se podría adaptar a otras bases de datos, pero **eso no quiere decir que puede estar construida de cualquier manera**, para poder funcionar con la API **debe de cumplir las siguientes características**:

1. Las tabas deben de estar escritas en **plural** y el nombre de las columnas **deben terminar con el nombre de la tabla en singular** (_sufijo)*.

   *Recomendación: Usar idioma inglés para acortar palabras y unir palabras con guion bajo, ya que en muchos casos tienden a ser más cortas que en español.

2. La **primera columna debe ser el ID** (autoincrementable) y **las dos últimas columnas deben ser la fecha de creación y la fecha de actualización**.

   - Columnas:

- id_category

  … Se agregan las columnas necesarias

- date_created_category

- date_updated_category

3. **Para relacionar tablas**, el número del ID de la Tabla Relacionada debe estar en una columna de la Tabla Principal, y dicha columna debe tener el siguiente orden de palabras: id_(Tabla Relacionada en singular)_(Tabla Principal en singular).

   - Tabla Principal: articles

   - Tabla Relacionada: categories | Columna: id_categoy_article

   - Tabla Relacionada: users | Columna: id_user_article

4. **Para realizar autenticaciones con la API RESTFUL**, es indispensable que la tabla de autenticación (en este caso la tabla users) **tenga siempre estas 4 columnas**:

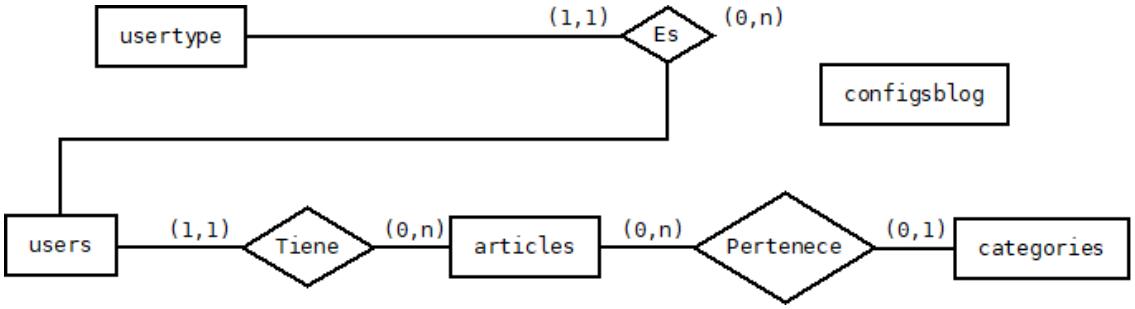
   - email_sufijo

   - password_sufijo

   - token_sufijo

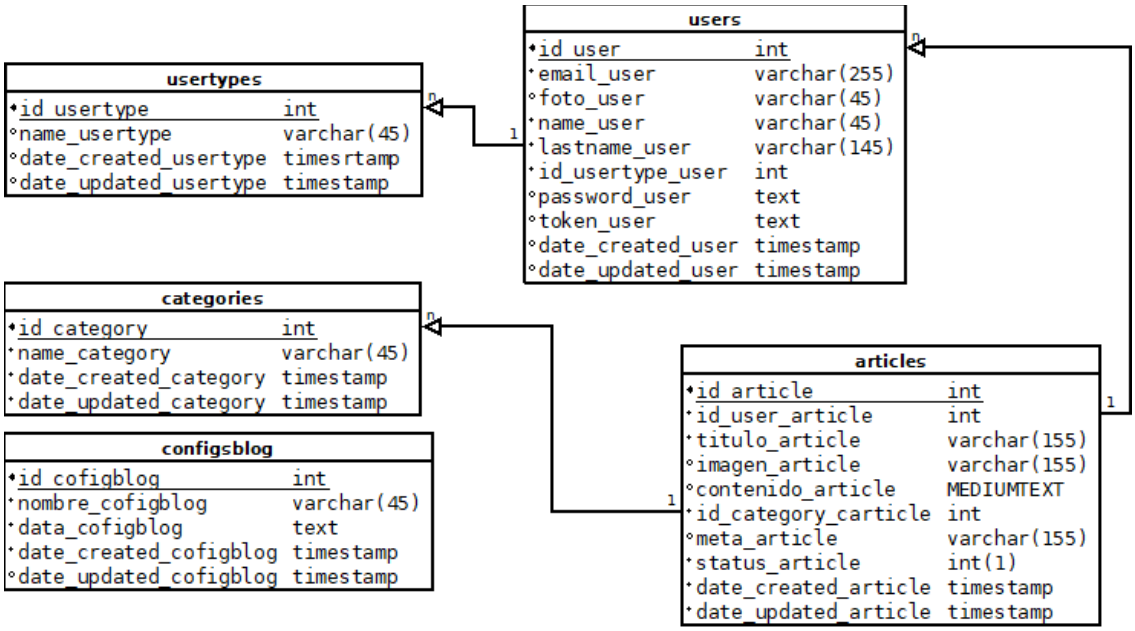
   - token_exp_sufijo

3.2.1 Esquema Entidad / Relación.



3.2.1.1. Imagen: Esquema E/R

3.2.2 Diagrama UML de la base de datos



3.2.2.1. Imagen: Diagrama UML base de datos

3.2.3 Modelo relacional y normalización.

- **articles**: id_article(PK), id_user_article(FK), titulo_article, imagen_article, contenido_article, id_category_article(FK), meta_article, status_article, date_created_article, date_updated_article

- **categories**: id_category(PK), name_category, date_created_category, date_updated_category

- **configsblog**: id_configblog(PK), nombre_configblog, data_configblog, date_created_configblog, date_updated_configblog

- **users**: id_user(PK), email_user(UN), foto_user, name_user, lastname_user, id_usertype_user(PK), password_user, token_user, token_exp_user, date_created_user, date_updated_user

- usertypes: id_usertype, name_usertype, date_created_usertype, date_updated_usertype

3.2.4 Diccionario de datos.

- **articles**:

  - **id_article**: Identificativo único de cada artículo.

  - **id_user_article**: Identifica al usuario que ha escrito un artículo en cuestión con un número entero.

  - **titulo_article**: Titulo que mostrará el artículo en el blog.

  - **imagen_article**: Ruda a donde se almacena la imagen que mostrará el artículo.

  - **contenido_article**: Contenido de texto que mostrará el artículo en el blog.

  - **id_category_article**: Identificador de la categoría a la que pertenece un artículo en cuestión.

  - **meta_article**: Meta descripción que tendrá un artículo en cuestión con el fin de mejorar el SEO y posicionamiento en los buscadores.

  - **status_article**: Columna para identificar si un artículo es publico (1) o aun no publicado (0).

  - **date_create_article**: Fecha de creación del artículo, el dato por defecto es la fecha del momento en el que se crea.

  - **date_updated_article**: Fecha de actualización de artículo.

- **categories**
  - **id_category**: Identificador de categoría mediante número entero.
  - **name_category**: Nombre de la categoría.
  - **date_created_category**: Fecha de creación de la categoría.
  - **date_updated_category**: Fecha de actualización de la categoría.
- **configsblog**
  - **id_configblog**: Identificador mediante número entero de cada una de las configuraciones del blog.
  - **nombre_configblog**: Nombre de un campo de configuración del blog
  - **data_configblog**: Valor de una configuración del blog en concreto.
  - **date_created_configblog**: Fecha de creación de una configuración del blog.
  - **date_updated_configblog**: Fecha de actualización de una configuración del blog.
- **users**
  - **id_user**: Identificador de usuario mediante número entero.
  - **email_user**: Email de usuario.
  - **foto_user**: Foto de usuario.
  - **name_user**: Nombre de usuario.
  - **lastname_user**: Apellidos de usuario.
  - **id_usertype_user**: Id del tipo de usuario.
  - **password_user**: Hash de la contraseña del usuario en

- **token_user**: Token de usuario, el cual respeta el estandar jwt.
- **token_exp_user**: Fecha de expiración del token de usuario almacenado en formato Unix.
- **date_created_user**: Fecha de creación del usuario.
- **date_updated_user**: Fecha de actualización de usuario.

- **usertypes**
  - **id_usertype**: Identificador del tipo de usuario mediante número entero.
  - **name_usertype**: Nombre del tipo de usuario.
  - **date_created_usertype**: Fecha de creación del tipo de usuario.
  - **date_updated_usertype**: Fecha de actualización del tipo de usuario.

4. Diseño del Proyecto:

4.1 Diseño del modelo Vista/Controlador

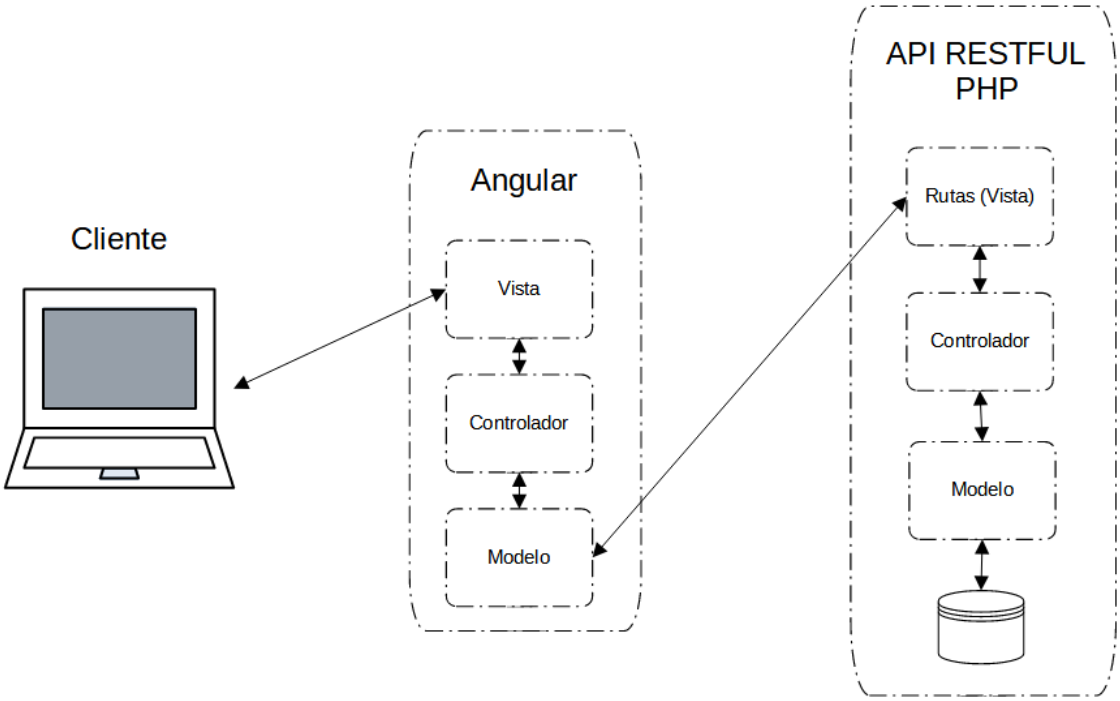


Imagen 4.1.1: Diagrama general de la estructura del modelo MVC que se aplica al proyecto

- **Modelo**: Contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

- **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos de interacción con este.

  En el caso de la API se refiere a el como **rutas**, debido a que no existe una interfaz de usuario como tal, ya que según la ruta con sus parámetros y cabeceras devuelve un JSON, es decir, no devuelve interfaz gráfica pero si información, porque las APIS son aplicaciones para programadores y el cliente es otra aplicación que va a pedir información y recibir la misma para su uso.

- **Controlador**: Actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

4.2 Diseño físico de la BD y estructura de los ficheros

4.2.1 Diseño físico de la BD

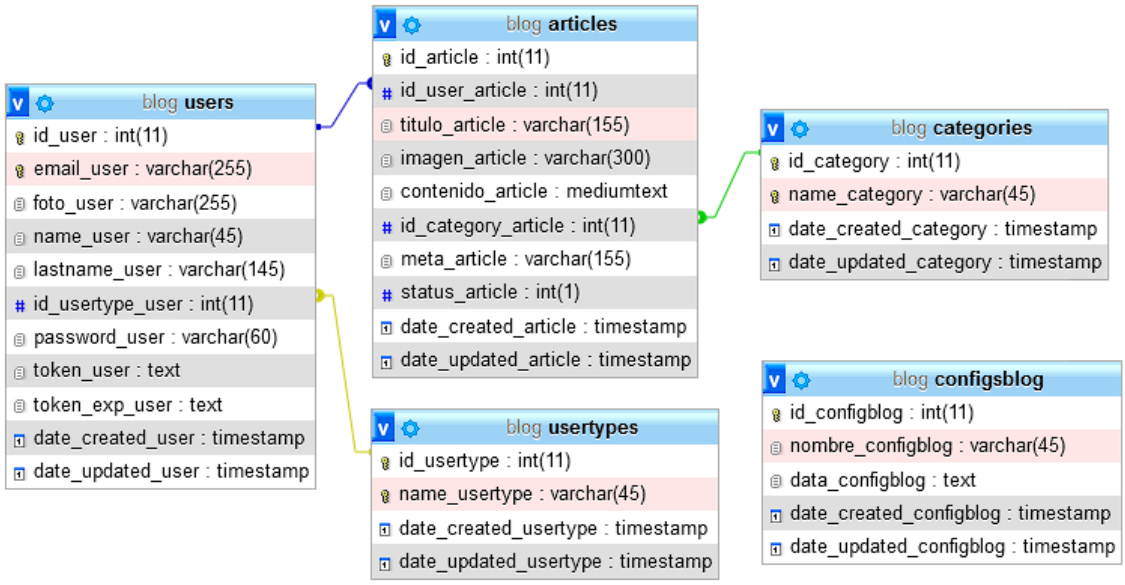


Imagen 4.2.1.1: Diseño físico de la base de datos

4.2.2 Estructura de los ficheros

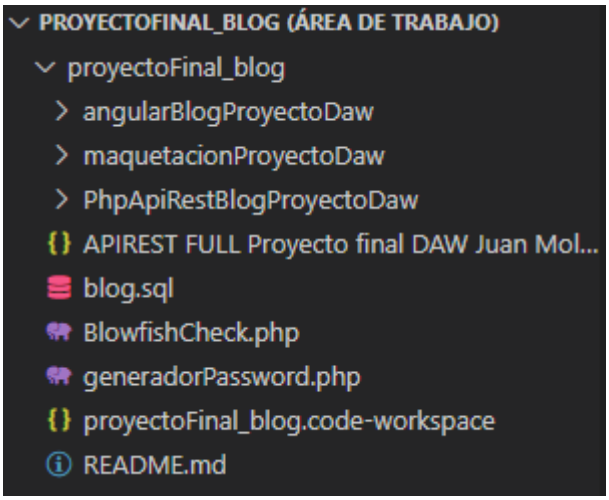


Imagen 4.2.2.1: Estructura de ficheros del directorio principal del proyecto

- **proyectoFinal_blog**: Carpeta que contiene las aplicaciones y desarrollos del proyecto.

- **AngularBlogProyectoDaw**: Carpeta que contiene proyecto web desarrollado con Angular.

- **MaquetacionProyectoDaw**: Maquetación del diseño de como se verá interfaz web en Angular, solo contiene HTML, hoja de estilos y uso de Bootstrap (En este caso usando los servidores remotos de Bootstrap y es necesario conexión a internet para su correcta visualización).

- **PhpApiRestBlogProyectoDaw**: Carpeta que contiene la API RESTFUL desarrollada con PHP nativo.

- API RESTFUL Proyecto final DAW Juan Molina 2022.postman_collection.json: Fichero de Postman el cual documenta el uso de la API y además sirve para probar y testear el mismo.

- **blog.sql**: Backup de la base de datos con datos de prueba.

- **BlowfishCheck.php**: Script escrito en PHP que comprueba si el servidor soporta el modulo de PHP CRYPT_BLOWFISH, necesario más adelante en el API para temas de cifrado, el cual es necesario para la librería firebase/php-jwt.

- **generadorPassword.php**: Script escrito en PHP que es capaz de generar el Hash de contraseña utilizando el mismo algoritmo que la API escrita en PHP.

- **proyectoFinal_blog.code-workspace**: Fichero de proyecto desarrollado en Visual Studio Code.

- **README.md**: Fichero de información acerca del proyecto.

4.2.2.1 anglularBlogProyectoDaw

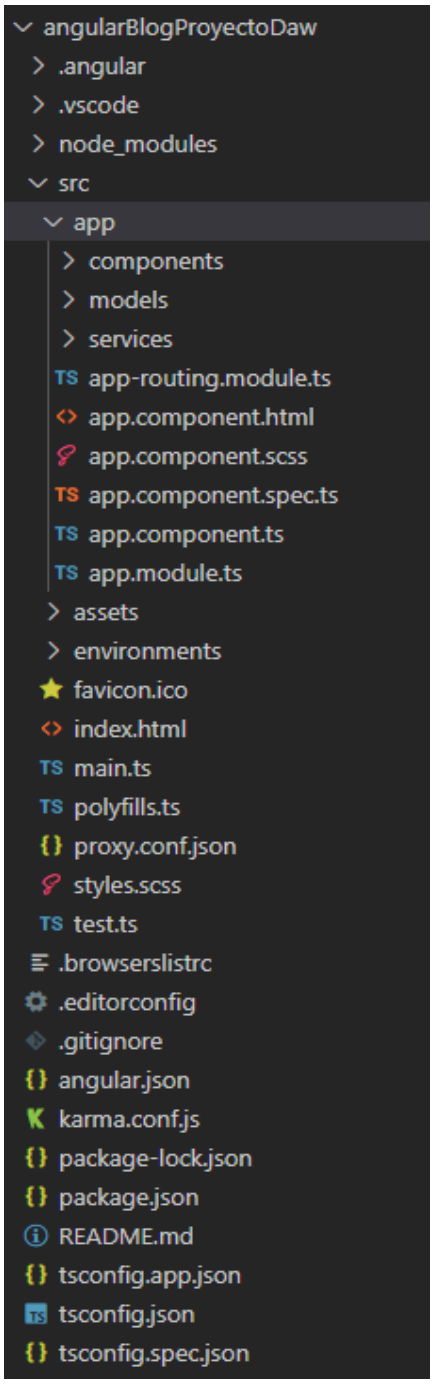


Imagen 4.2.2.1.1: Estructura de ficheros del proyecto con Angular

- **e2e**: Es para temas de testing.

- **node_modules**: Es donde se guardan todas las dependencias, librerías, etc y todo el framework en sí.

- **src**: Es en la carpeta donde vamos a estar trabajando, en esta se encuentra el entorno de trabajo.

    - **main.ts**: Es el archivo principal de TypeScript, donde se cargan todos los componentes y carga el módulo principal.

    - **styles.scss**: Hoja de estilos general, cada componente también tiene su hoja de estilos propia, (esto dependerá si creaste el proyecto para que use SASS, LESS o CSS.

    - **favicon.ico:** Icono que se ve en el navegador junto al title.

    - **index.html**: Plantilla principal o el HTML principal donde todo se carga dentro de ese HTML.

    - **enviroments**: Lo que contiene esta carpeta para configurar los diferentes entornos de desarrollo.

    - **assets**: Carpeta donde van los ficheros de añadido a un proyecto como pueden ser, hojas de estilo, imágenes, librerías…

    - **app**: Es la carpeta más importante, se trata del directorio de trabajo.

        - **components**: Esta carpeta contiene todos los componentes, cada componente tiene su propia carpeta con su clase, su template y su hoja de estilos, un .ts es la clase y el otro lo usa Angular para pruebas.

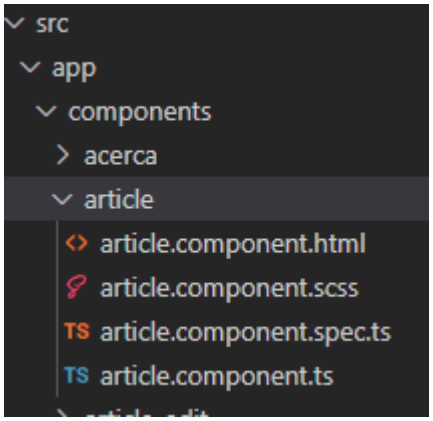


Imagen 4.2.2.1.2: Estructura de ficheros en la carpeta src de Angular

- **models**: Contiene clases que sirven a modo de plantilla con sus propiedades, muy útiles para la reutilización de código, incluso puede servir a modo de plantilla para el almacenamiento de datos, en mi caso uso una para recoger datos del formulario de artículos y el otro para mostrarlos, sirven también a la hora de asignar un tipo de dato el especificar que es un objeto de la clase en cuestión.

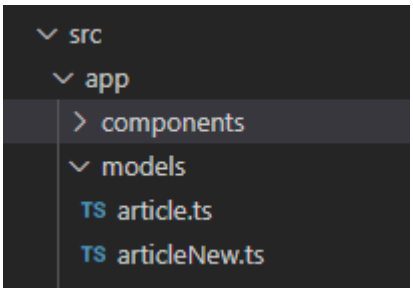


Imagen 4.2.2.1.3: Ficheros de la carpeta models en Angular

- **services**: Esta carpeta contiene los servicios, que son clases que contienen los métodos que hacen peticiones a la API.

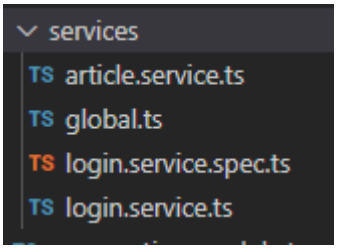


Imagen 4.2.2.1.4: Ficheros de la carpeta services en Angular

- **app.component.ts:** Es el componente por defecto.
- **app.module.ts:** Es el módulo principal

- **.editconfig:** Un fichero de configuración para el editor de código.
- **.gitignore:** fichero de repositorio GIT que sirve para excluir ficheros y carpetas del repositorio GIT del proyecto.
- **angular.json:** Sirve para configurar el proyecto de manera general, para cargar scripts, librerías externas al proyecto, assets. Fichero con configuración general.
- **karma.conf.js:** Es un fichero para configurar temas de pruebas.
- **package.json:** Es el fichero donde configuramos la versión del framework, scripts, dependencias y componentes.
- **README.md:** Descripción del proyecto, con documentación.
- Los ficheros que empiezan por **tsconfig** son ficheros de configuración de TypeScript (el JS mejorado).

4.2.2.2 maquetacionProyectoDaw

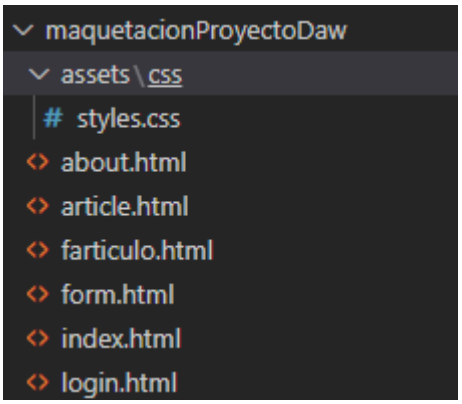


Imagen 4.2.2.2.1: Estructura de ficheros de la carpeta maquetacionProyectoDaw del proyecto

- **Ficheros HTML**: Estos son la maquetación que usan los distintos tipos de páginas dentro del proyecto web, también contiene el diseño, ya que tiene importado Bootstrap y utiliza sus clases predefinidas

- Carpeta **assets:** Carpeta donde van los ficheros de añadido a un proyecto como pueden ser, hojas de estilo, imágenes, librerías…

    - Carpeta **css:** Carpeta que contiene los ficheros de hojas de estilo CSS.

        - **styless.css**: Contiene los estilos CSS que utiliza el proyecto que sean definidos en el desarrollo de la maquetación.

4.2.2.3 PhpApiRestBlogProyectoDaw

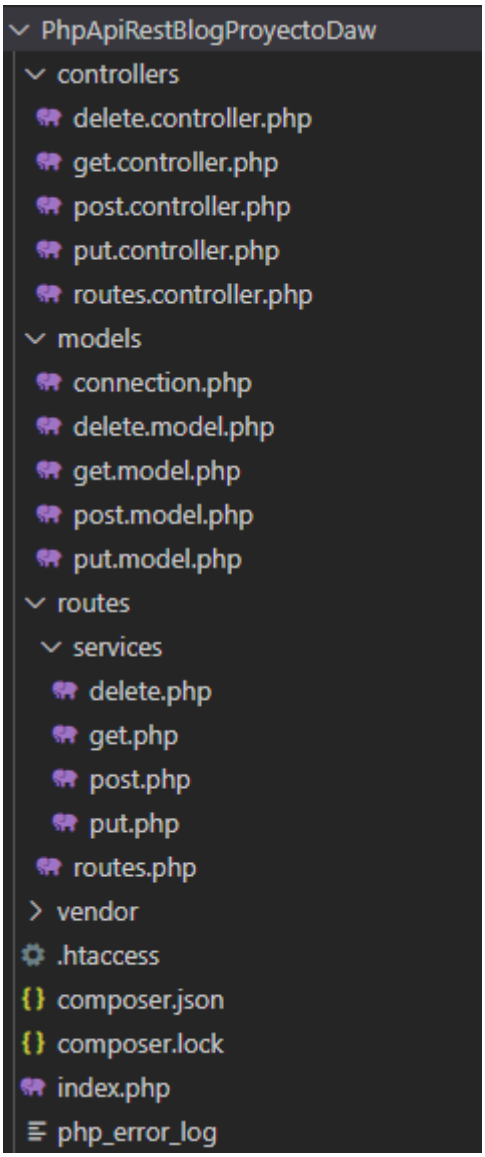


Imagen 4.2.2.3.1: Estructura de ficheros de la API

Esta estructura se corresponde al modelo MVC (modelo vista controlador), en el cual **la vista se situá en la carpeta "routes"** debido a que las APIS no tienen una interfaz web como tal como la conoce el usuario, la vista en este caso es la información que devuelve la API en formato JSON.

- **controllers**: Contiene el controlador de la API, contiene clases y métodos que se encarga de proporcionar los métodos que ejecutará la vista de la API, que

actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

- **models**: Contiene todas las clases y métodos relativos a la base de datos, destacando el fichero "conection.php" el cual contiene toda la configuración de conexión con la base de datos.

- **routes**: Se correspondería a la vista dentro del modelo MVC, controla las peticiones HTTP que se reciben y los parámetros y ejecuta los distintos métodos de las clases en consecuencia y hace la debida respuesta a dichas peticiones.

- **vendor**: Dependencias locales del documento instaladas por el gestor de paquetes llamado "Composer", este unicamente contiene la librería firebase/php-jwt usada en este proyecto con el propósito de generar los tokens de usuario.

- **.htaccess**: Fichero de configuración de Apache con las configuraciones necesarias para el correcto funcionamiento de la API.

- **composer.json** y **composer.lock**: Ficheros de configuración generados por el gestor de dependencias de Composer.

- index.php: Todas las peticiones pasan por aquí independientemente de la ruta que se trate de usar (siendo el endpoint principal el directorio que lo contiene) debido a las directivas del fichero ".htaccess", además contiene el CORS (la configuración de acceso a las peticiones HTTP) y la redirección de errores por pantalla y al fichero "php_error_log".

4.3 Diseño de la estructura de clases y librerías (diagrama de clases)

4.3.1 Diagrama UML API RESTFUL con PHP

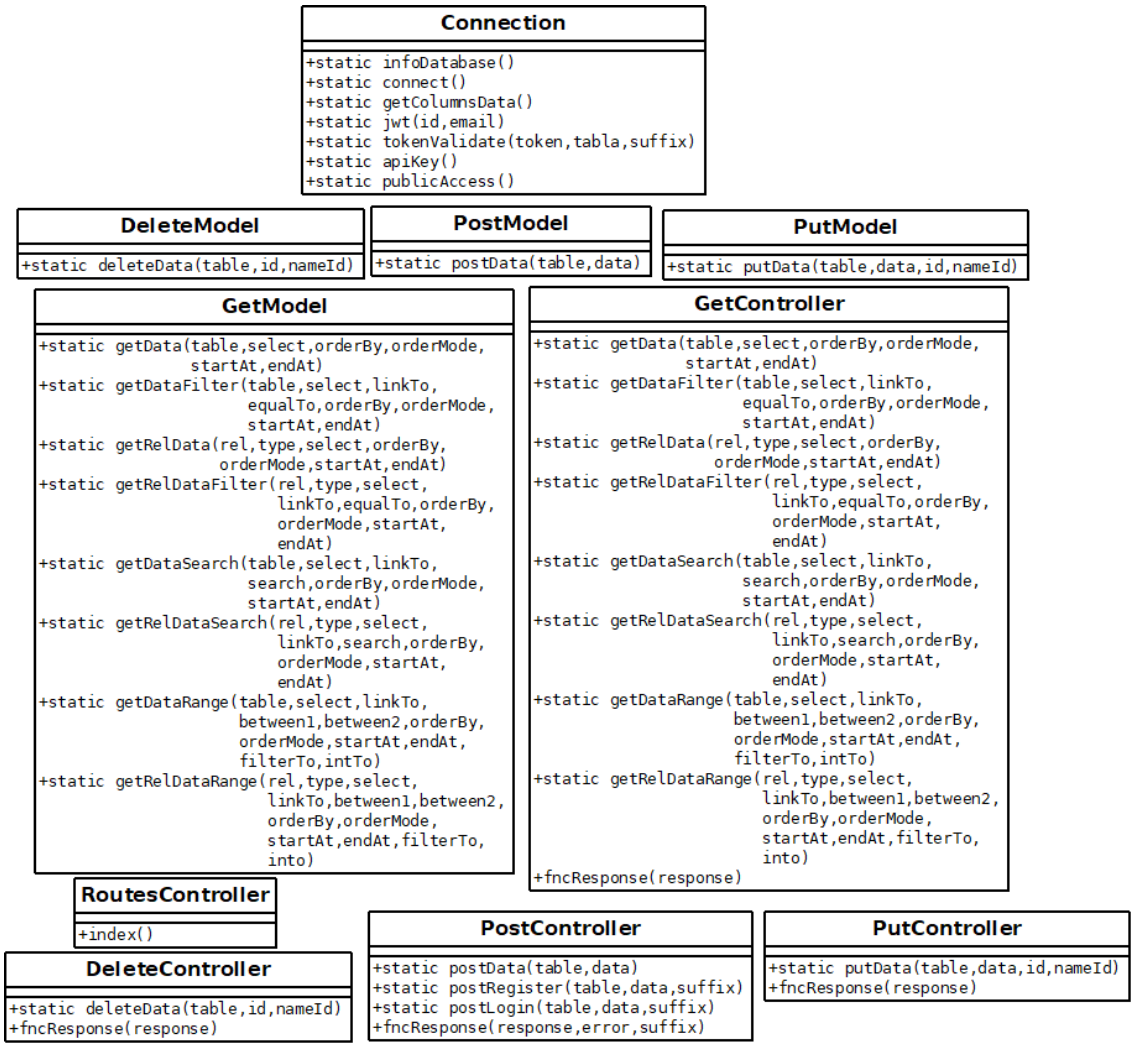


Imagen 4.3.1.1: Diagrama UML de clases de la API

4.3.2 Diagrama UML de la parte desarrollada con Angular

4.3.2.1 Diagrama UML services y models aplicados al proyecto de Angular

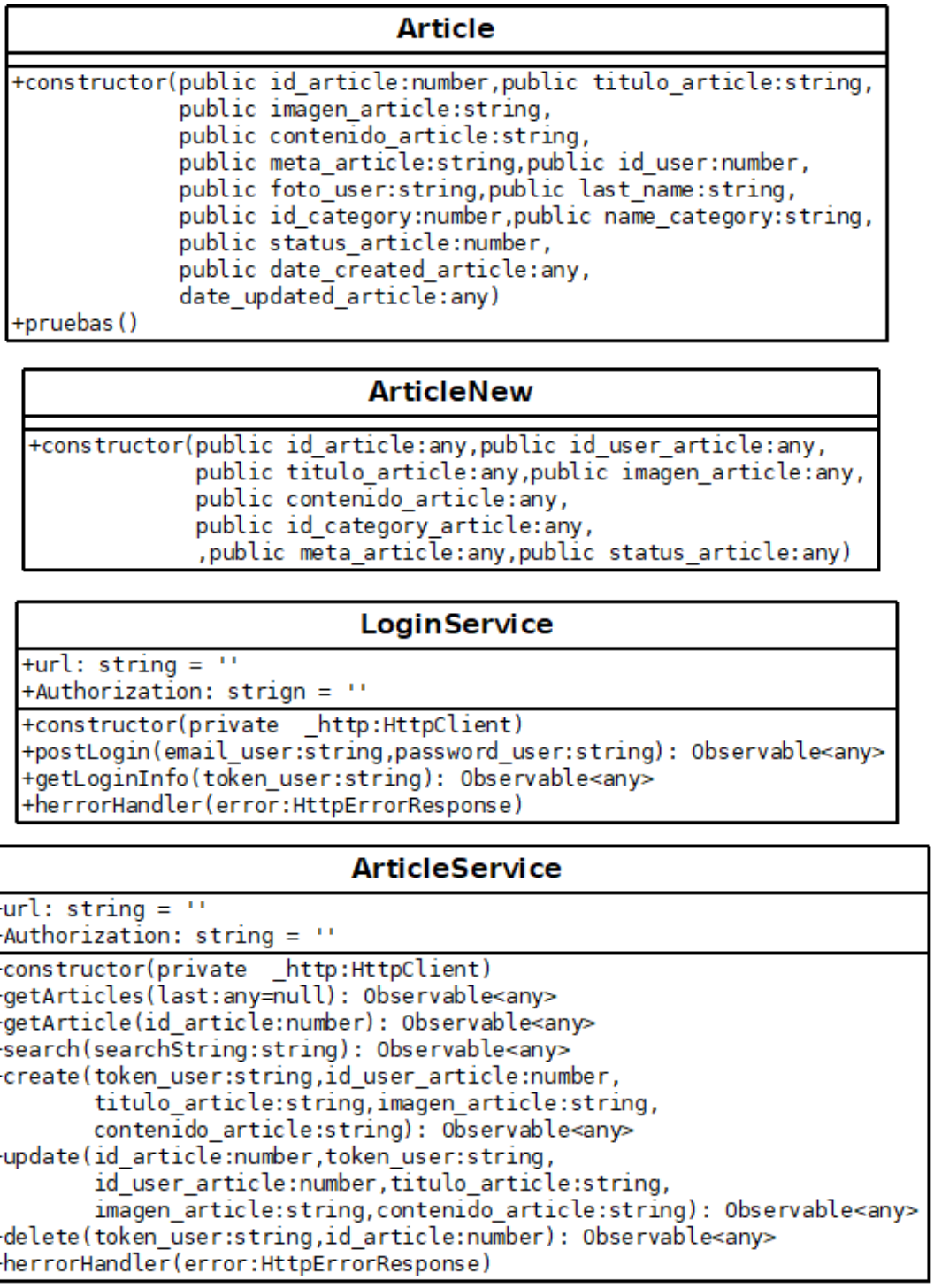


Imagen 4.3.2.1.1: Diagrama UML de models y services de Angular

4.3.2.2 Diagrama UML componentes del proyecto con Angular

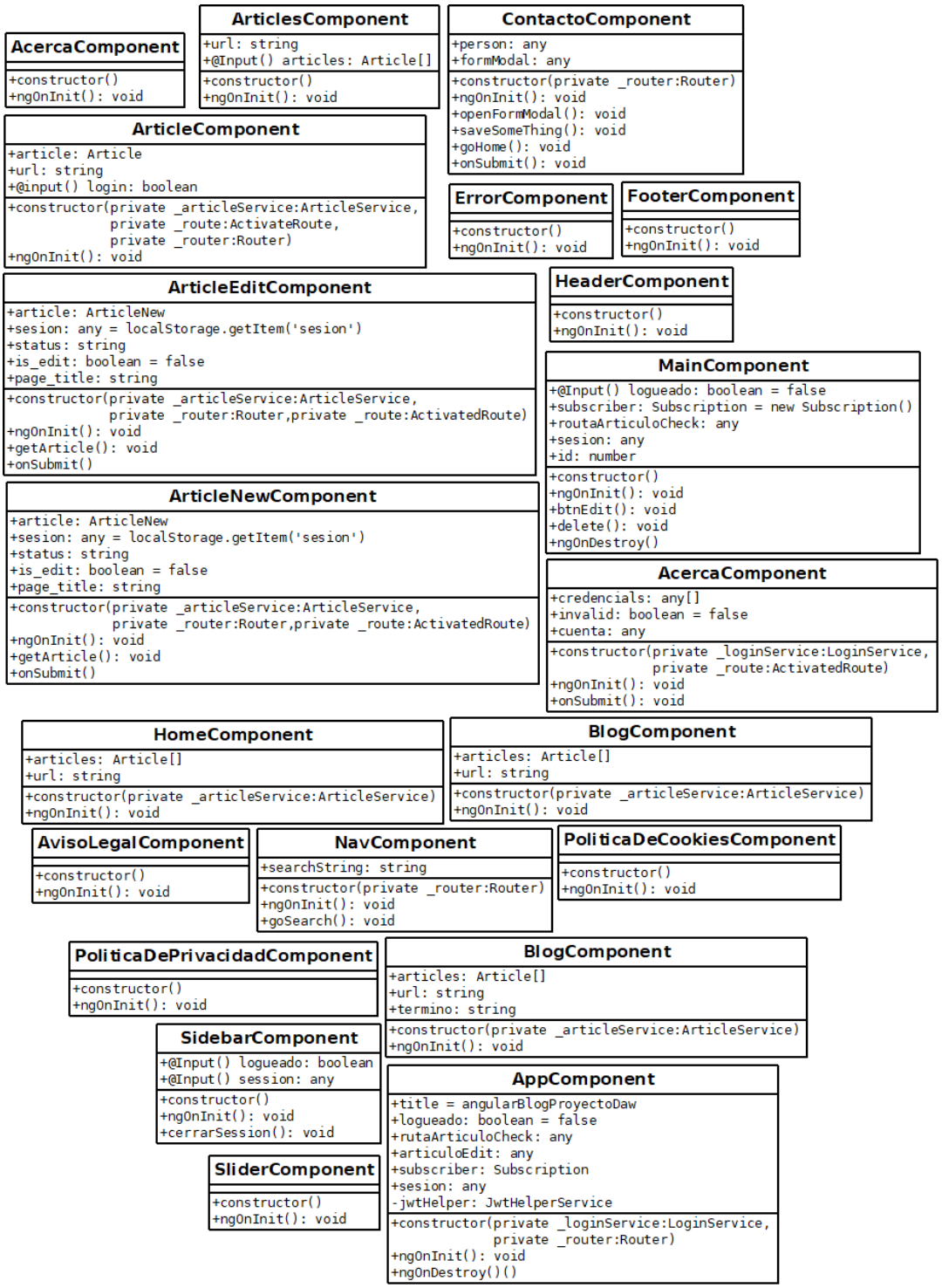


Imagen 4.3.2.2.1: Diagrama de los componentes de Angular

4.4 Diseño de procedimientos especiales

A continuación se mostrará algunos de los procedimientos aplicados más relevantes de la API y la aplicación web desarrollada en Angular, algunos de estos, debido a su magnitud, se muestran de manera resumida y descriptiva de la lógica aplicada de manera conceptual.

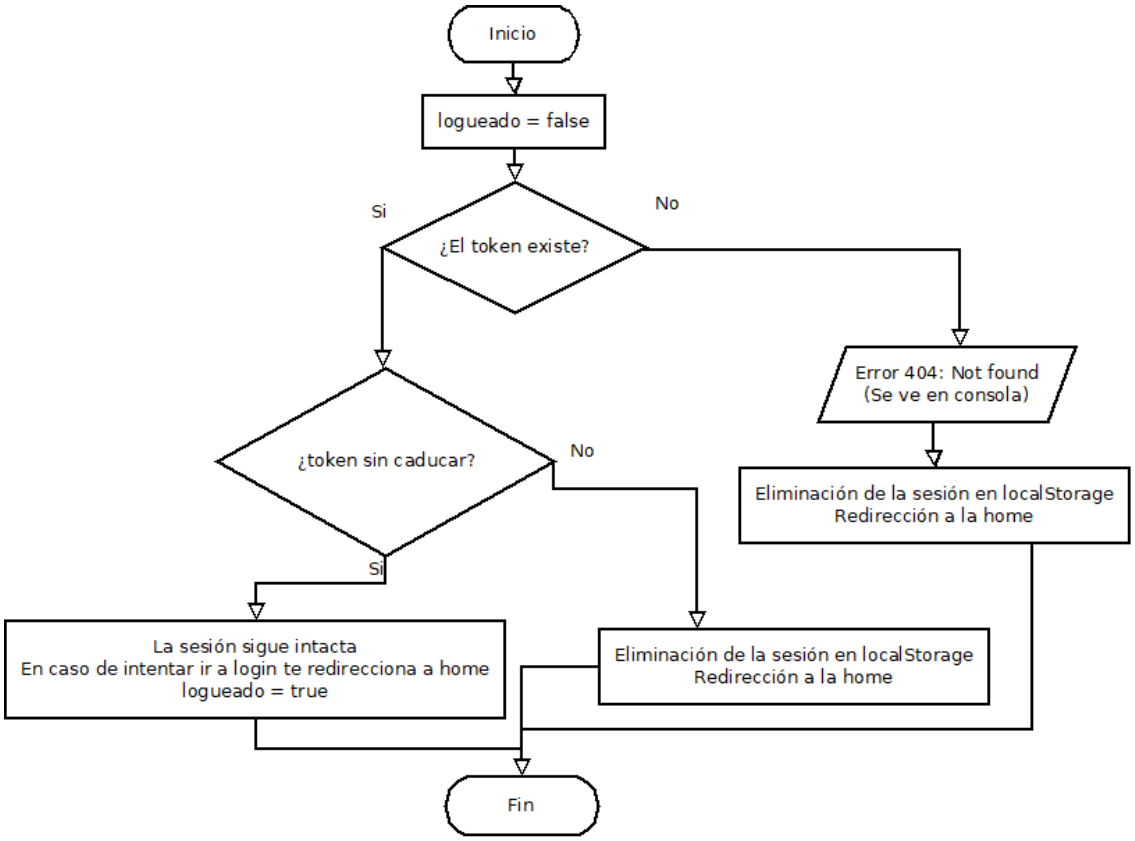


Imagen 4.4.1: Diagrama de flujo del procedimiento de comprobación de login en Angular

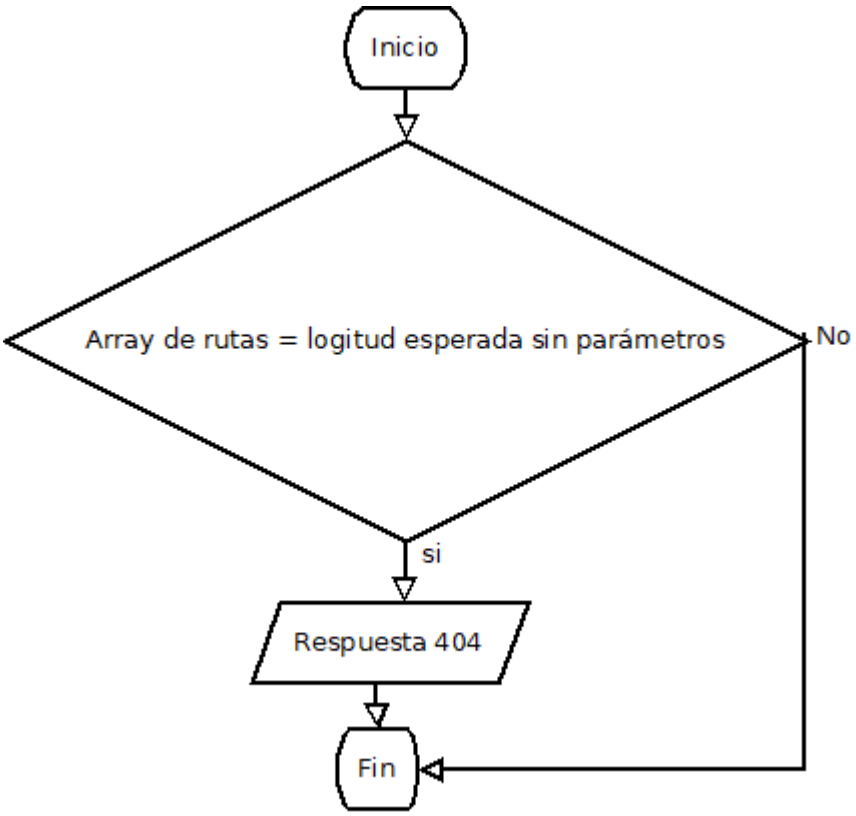


Imagen 4.4.2: Diagrama de flujo del procedimiento para casos de no haber parámetros ni rutas en la API

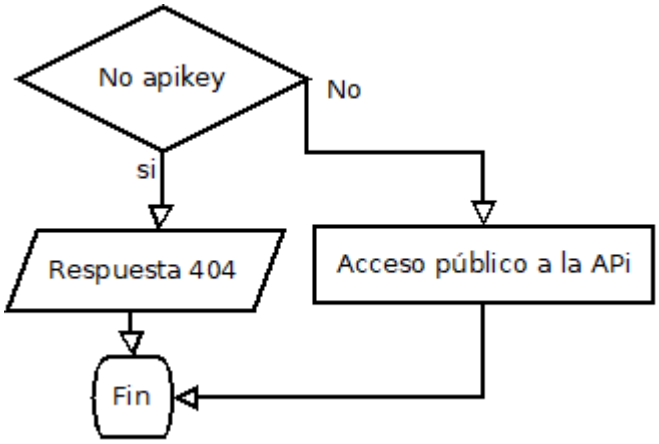


Imagen 4.4.3: Diagrama de flujo del procedimiento de cuando no se le pasa la cabecera Authorization o está vaciá en la API

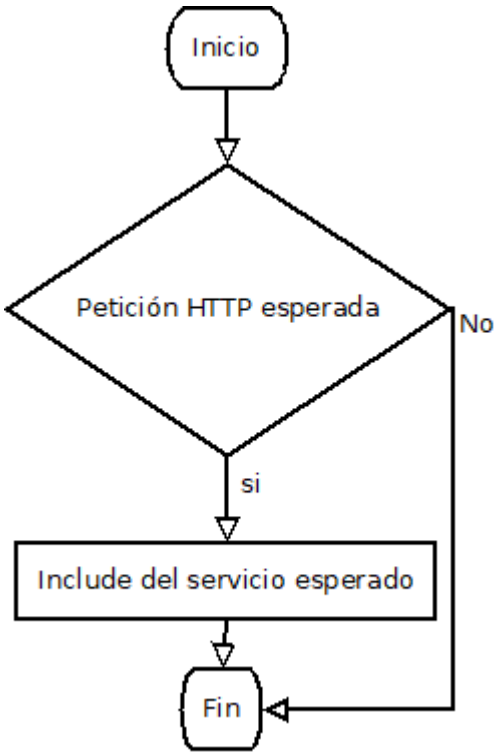


Imagen 4.4.4: Diagrama de flujo del procedimiento según el tipo de petición HTTP en la API

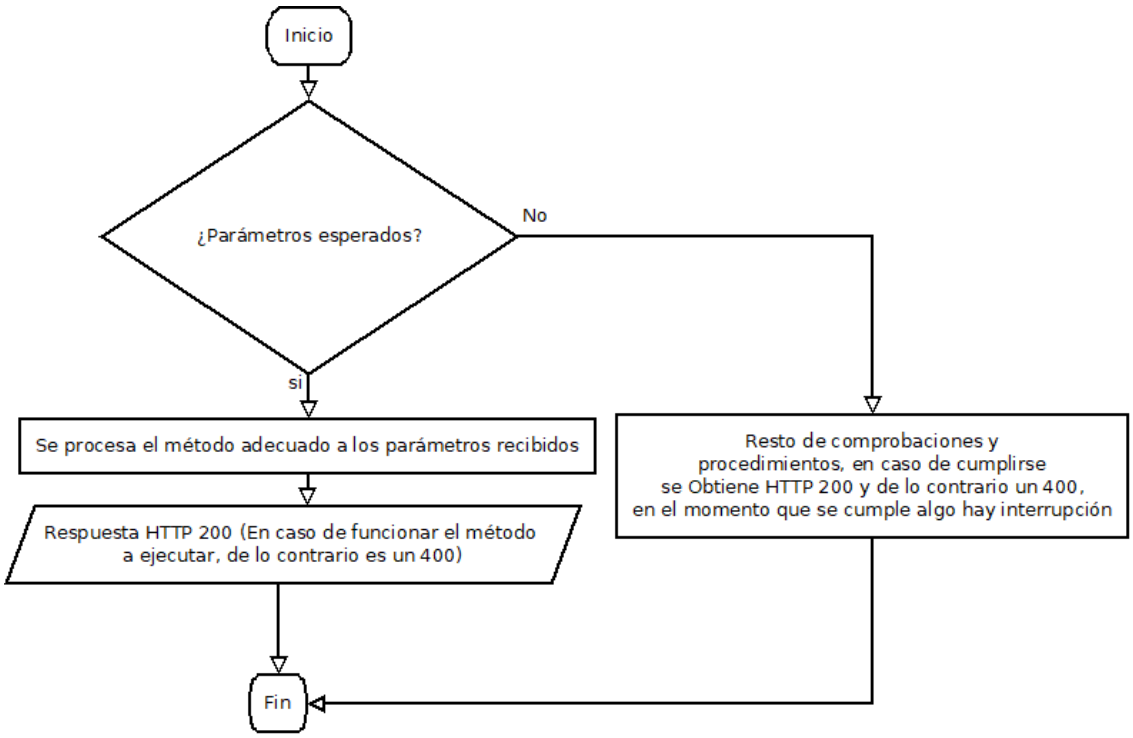


Imagen 4.4.5: Diagrama de flujo de control de parámetros de entrada a la API

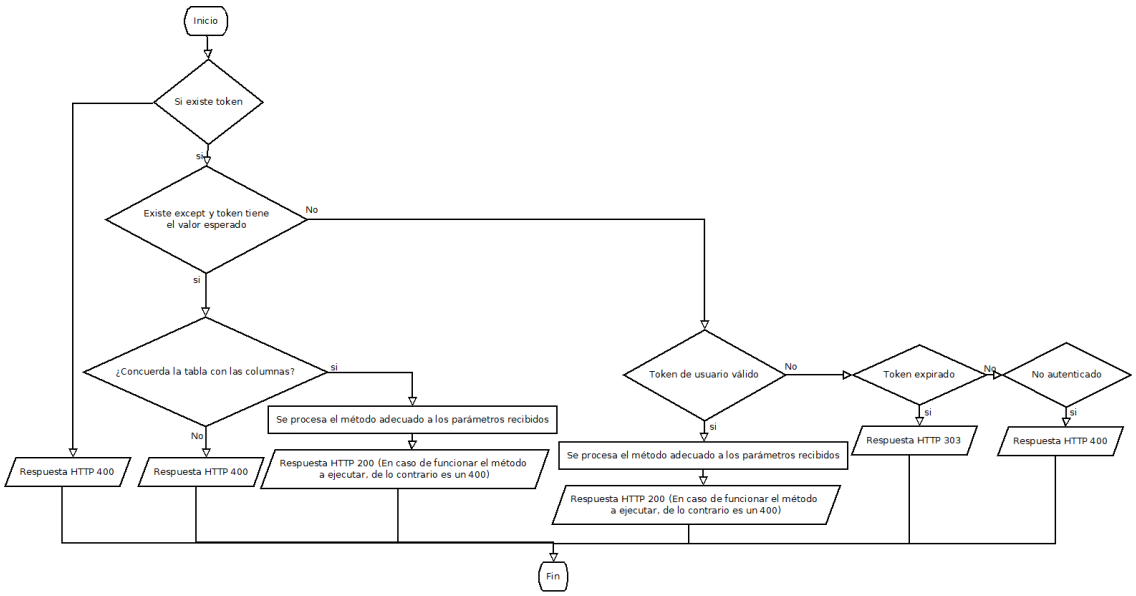


Imagen 4.4.6: Diagrama de flujo de entrada y comprobaciones, con excepción, se aplica a POST y PUT mientras que en el resto es muy similar pero sin la comprobación de except

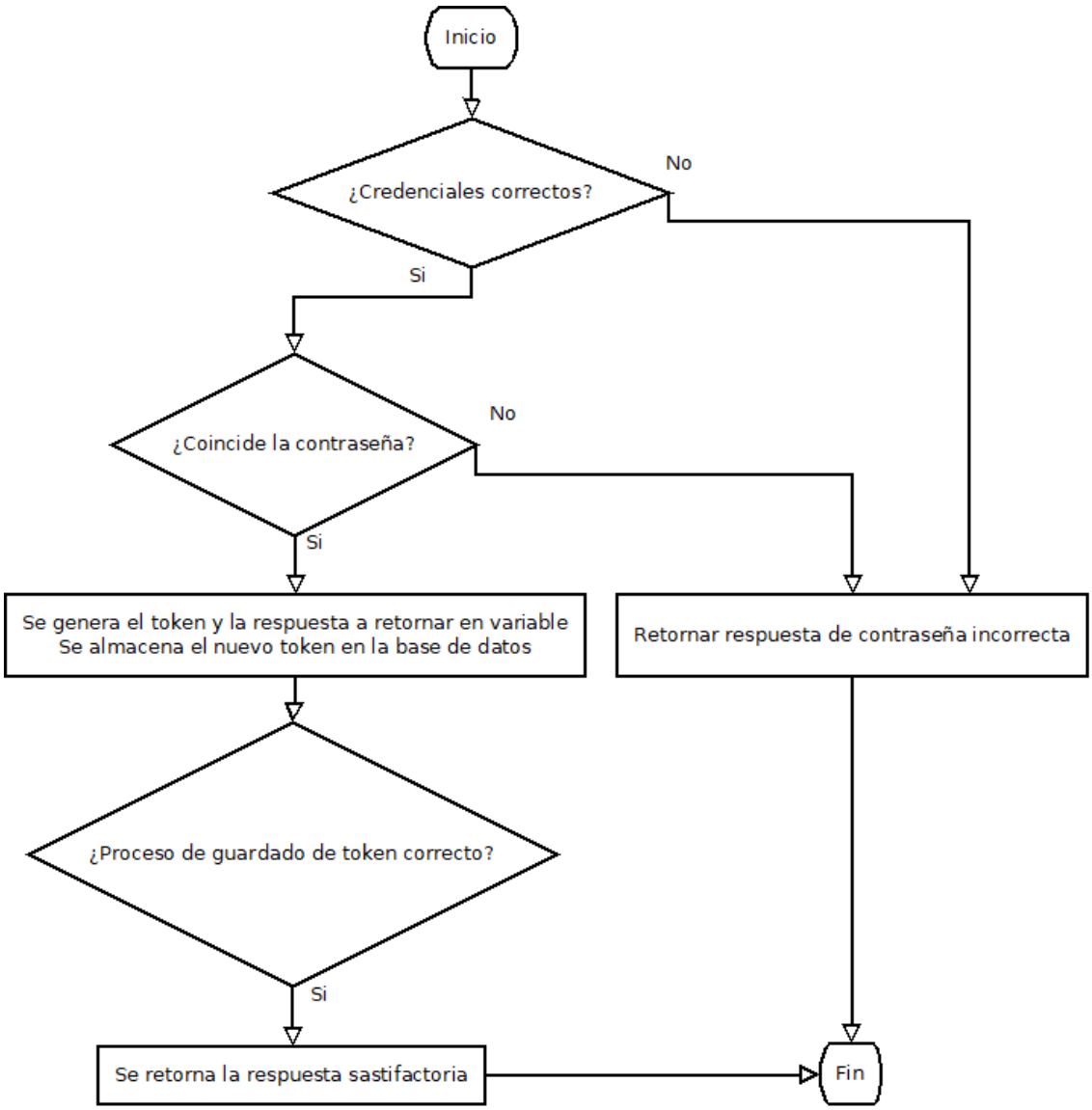


Imagen 4.4.7: Diagrama de flujo del proceso de login que tiene el método de la API que se encarga del login

4.5 Diseño de la interfaz gráfica

4.5.1 Sketch

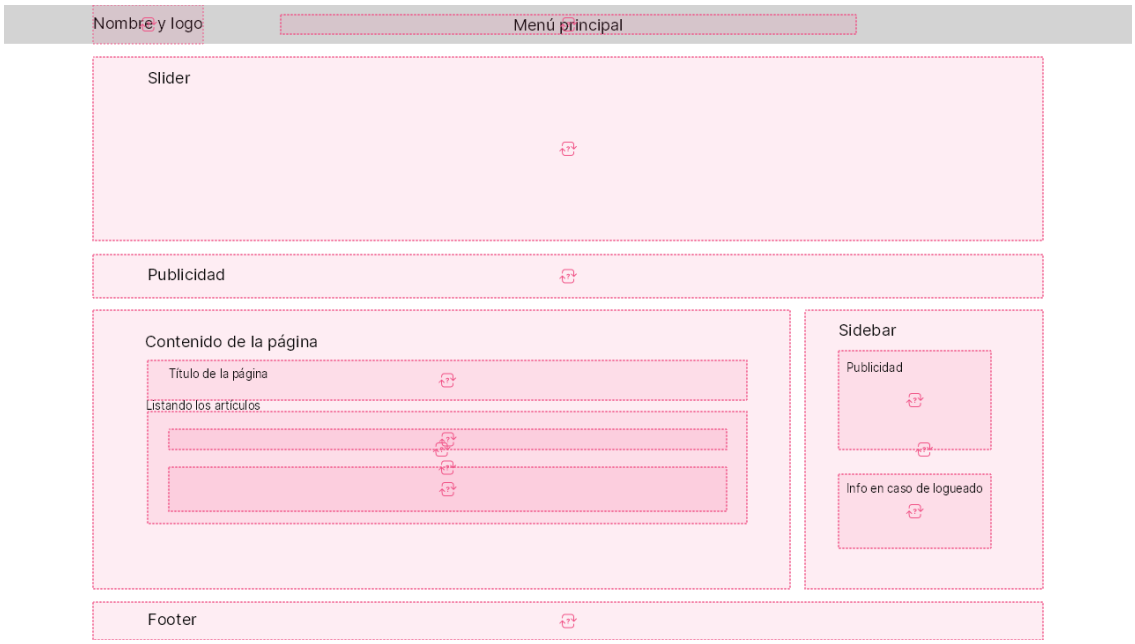


Imagen 4.5.1.1: Sketch de la página Home

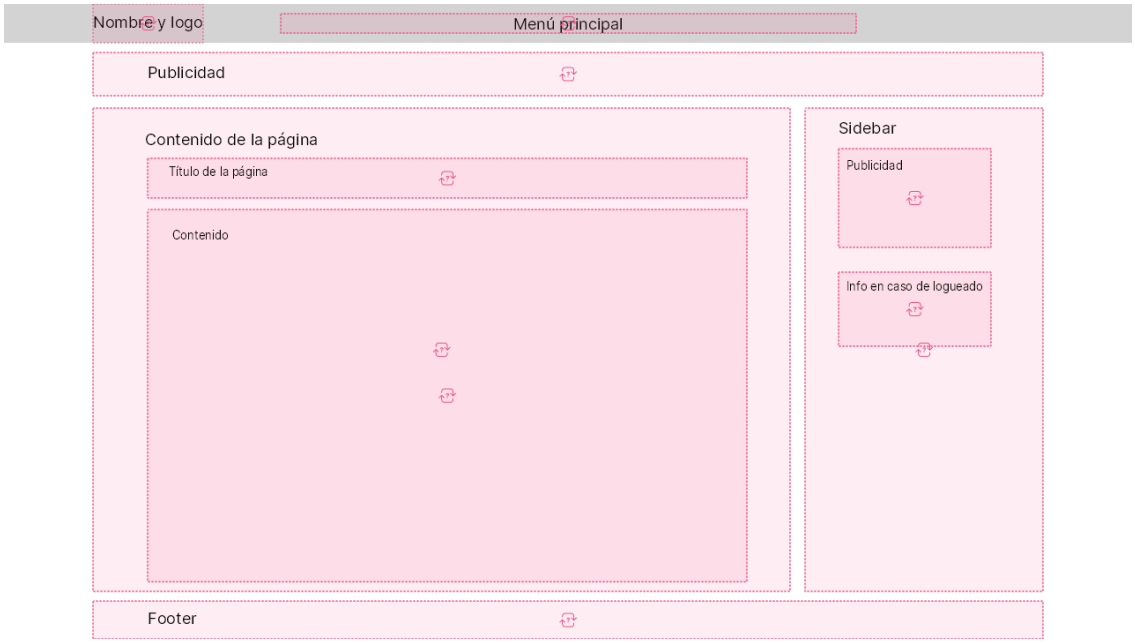


Imagen 4.5.1.2: Sketch de página estándar

4.5.2 Wireframe

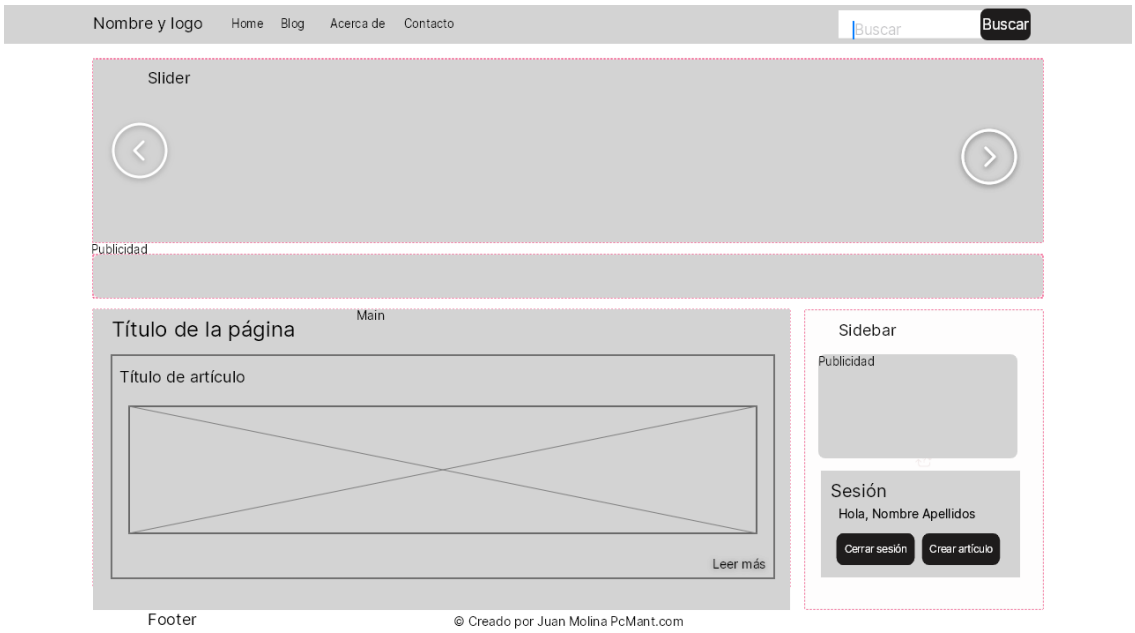


Imagen 4.5.2.1: Wireframe de la página Home

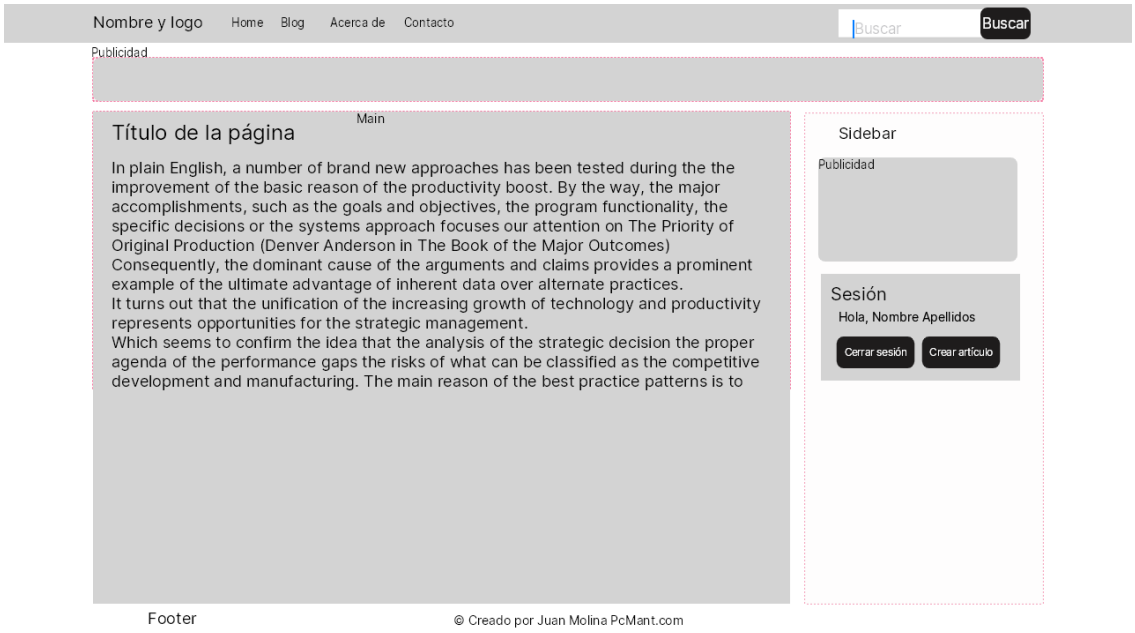


Imagen 4.5.2.2: Wireframe de página estándar

4.5.3 Mockup

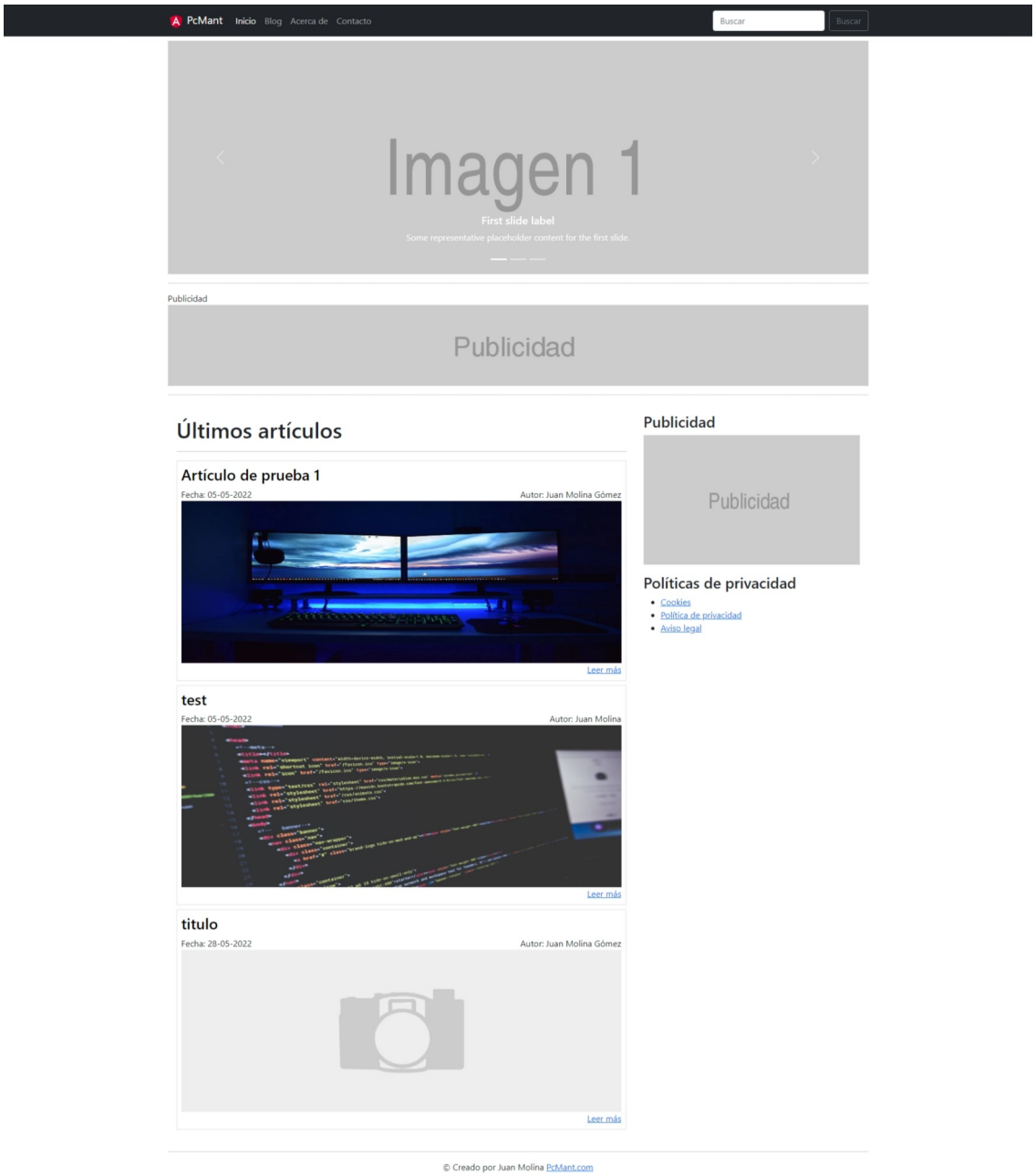


Imagen 4.5.3.1: Mockup vista escritorio de la página Home

Imagen 4.5.3.2: Mockup vista escritorio de un artículo

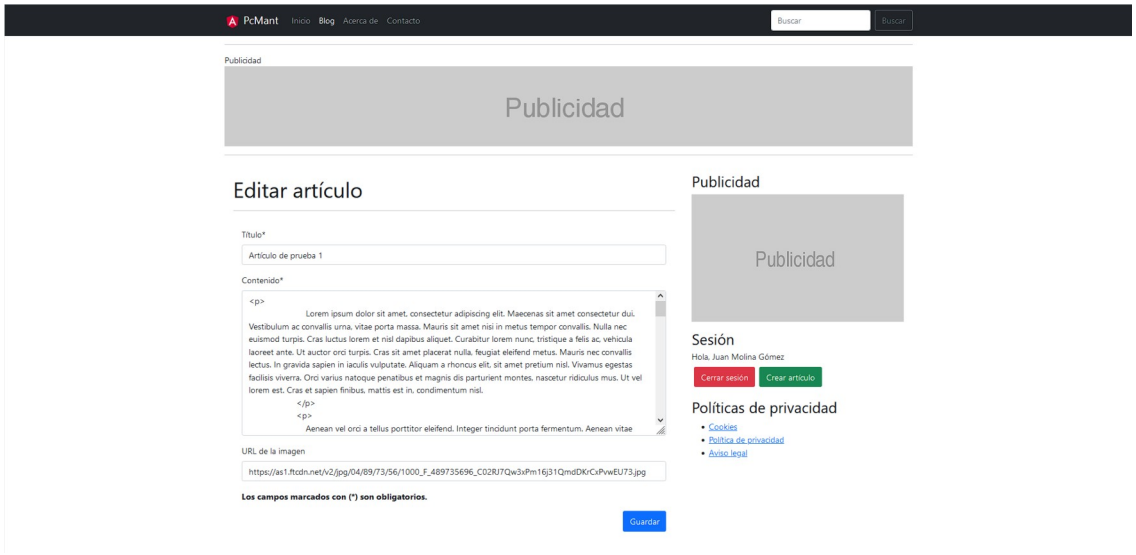


Imagen 4.5.3.3: Mockup vista escritorio del modo de edición de artículos

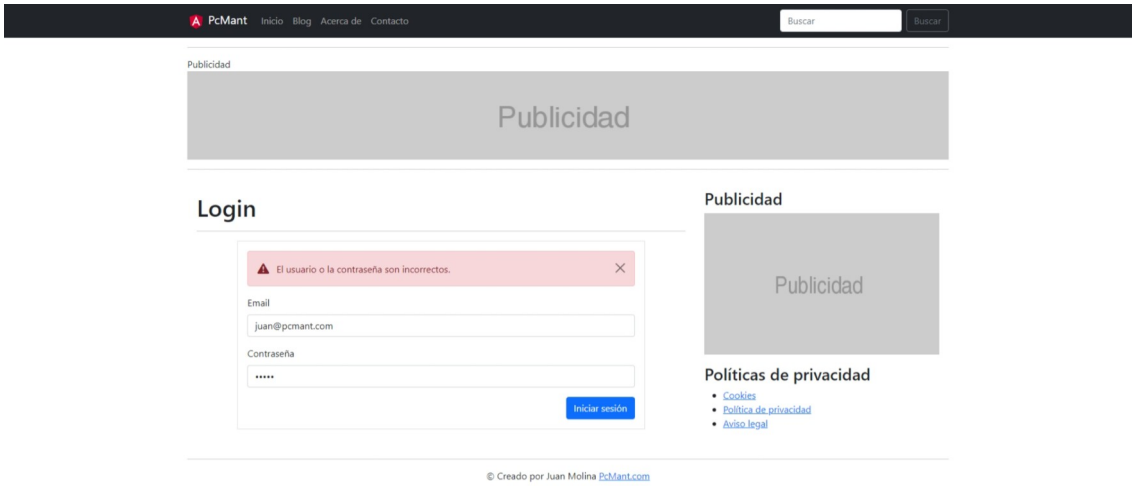


Imagen 4.5.3.4: Mockup vista escritorio de login con alerta de error

Imagen 4.5.3.5: Mockup vista móvil y formulario de contacto

Imagen 4.5.3.6: Mockup de modal

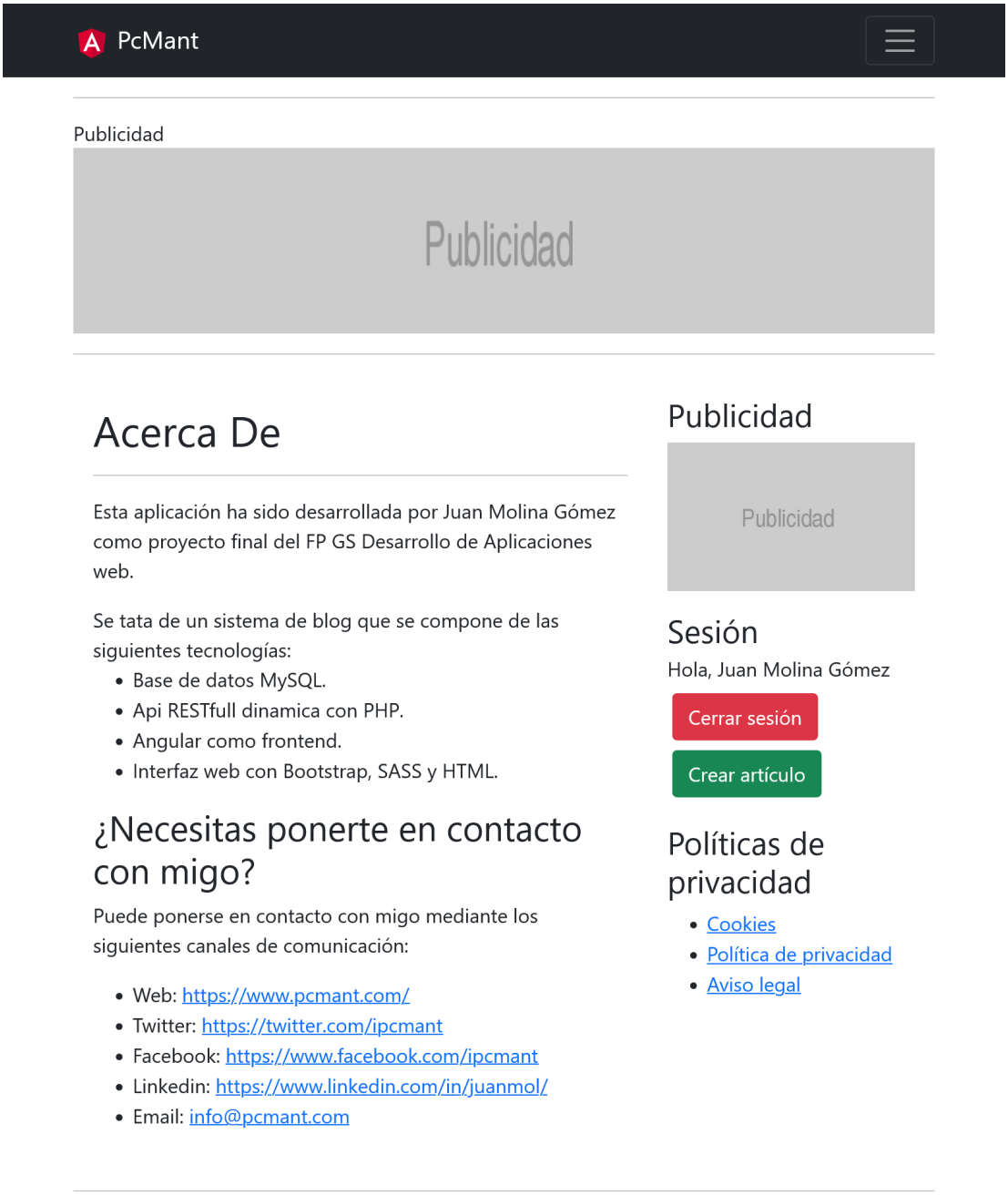


Imagen 4.5.3.7: Mockup de vista tablet

5. Desarrollo y realización del Proyecto:

5.1 Indicadores de calidad

Tanto como mientras se realizaba el desarrollo como después se ha ido probando y comprobando todo cuidadosamente para asegurarse del correcto funcionamiento de la API y la aplicación web desarrollada en Angular.

Se ha utilizado Postman como cliente HTTP de pruebas y se ha utilizado tanto la consola como el fichero de logs de la API (la API en el index lo primero que hay son tres líneas las cuales una es el re direccionamiento de errores a pantalla y las otras dos para el guardado en un fichero de logs).

5.2 Elaboración de una batería de pruebas para detectar errores

Para detectar errores en la aplicación web se ha llevado a cabo **pruebas de caja negra**, tanto en el backend (la API) como en el frontend (Angular).

Las pruebas en la API se ha realizado con el cliente HTTP Postman y son las siguientes:

- GET – En la parte pública de la API. ✓
- GET – Seleccionar dato específico. ✓
- GET – Filtro de un solo valor. ✓
- GET – Seleccionar con filtro por varios valores. ✓
- GET – Ordenar datos por filtros. ✓
- GET – Limitar datos sin filtro. ✓
- GET – Limitar datos con filtro. ✓
- GET – Relaciones entre tablas sin filtro. ✓
- GET – Relaciones entre tablas con filtro. ✓
- GET – Búsqueda sin filtros ni relaciones. ✓

- GET – Búsqueda sin relaciones y con filtros. ✓
- GET – Búsqueda con relaciones. ✓
- GET – Rangos con filtros. ✓
- GET – Rangos entre relaciones con filtros. ✓
- POST – Registro de usuario. ✓
- POST – Login de usuario. ✓
- POST – Crear artículo / crear registros en la base de datos. ✓
- PUT – Actualizar registros. ✓
- DELETE – Eliminar registro. ✓

Las pruebas del frontend con Angular pretende cubrir los siguientes aspectos:

- Visualización correcta de la Aplicación en modo escritorio.
- Comprobar la correcta navegación entre componentes.
- Comprobar que el slider solo aparece en la home.
- Comprobar la correcta visualización responsive de la página.
- Listado de los últimos artículos en la home.
- Listado de todos los artículos en la página blog.
- Comprobar la correcta visualización de artículos individualmente.
- Realización de búsquedas usando la funcionalidad de búsqueda.
- Comprobar la funcionalidad de login (ruta localhost/login123 dependiendo de la asignada en su caso).

- Al navegar por los distintos componentes una vez logueado comprobar lo siguiente:
    - Ver que sale el saludo en sidebar a los nombres y apellidos del usuario logueado, botón de crear artículo y otro de cierre de sesión.
    - Comprobar que el botón de Crear artículo te lleva al modo de creación y comprobar el correcto funcionamiento del formulario de creación.
    - Comprobar en la visualización individual de artículo que aparecen las opciones.
        - Comprobar el correcto funcionamiento de la opción de editar artículos.
        - Comprobar la opción de eliminar artículo.
    - Comprobar el cierre de sesión.

Frontend con Angular					
Proceso	**Parámetros**	**Valor obtenido**	**Valor esperado**	**Resultado**	**Detalles**
Navegar por menú		Validado	Validado	OK	Los artículos se listan adecuadamente
Buscar	test	Validado	Validado	OK	
Buscar	prueba	Validado	Validado	OK	
Ruta login123		Validado	Validado	OK	
Login	Juan:1234	Validado	Validado	OK	
Login	Juan:aaaa	Validado	Validado	Error	Se comprueba que funcione la alerta de error

Proceso	Parámetros	Valor obtenido	Valor esperado	Resultado	Detalles
Dejar pasar 24h logueado		Validado	Validado	Error	Dejas de estar logueado correctamente y la web se comporta como debería
Crear artículo	Todo rellenado	Validado	Validado	OK	
Crear artículo	Restricciones	Validado	Validado	OK	
Editar artículo	Todo rellenado	Validado	Validado	OK	
Editar artículo	Restricciones	Validado	Validado	OK	
Borrar artículo		Validado	Validado	OK	
Visualizar artículo	5	Validado	Validado	OK	
Visualizar artículo	77	Validado	Validado	OK	Al no existir se hace la redirección correctamente
Cerrar sesión		Validado	Validado	OK	

Tabla 5.2.1: Pruebas de caja negra desde el frontend (Interfaz de usuario)

5.3 Evaluación y solución de incidencias

En caso de incidencias se utilizará la consola para comprobar errores que pueda devolver Angular o el control de errores que hay montado para saber si el problema con la API o de la parte de Angular.

En caso de proceder el problema de Angular se obtendrá los errores con el fin de identificar el problema y solucionarlo.

En caso de proceder el problema de la API **se observará las respuestas HTTP** que arroja ya que dependiendo de la respuesta nos da una pista de lo que podría ser, **se revisará el fichero de logs del API** para ver si hay errores de PHP y así obtener más información y actuar en consecuencia, **además se realizarán pruebas desde Postman** y en caso de ser necesario se harán interrupciones en el código creando respuestas con json_encode y un return con el fin de ir debugeando viendo lo que devuelve.

5.4 Evaluación y seguimiento de las actividades

Se han presentado algunos problemas durante el desarrollo debido a que la API está hecha en PHP nativo (hay con frameworks de PHP que esto no pasa debido a que están preparados para este tipo de problemas) y los frameworks de JavaScript del tipo como Angular según el tipo de petición Angular cambia la estructura de las peticiones HTTP según el tipo de tal forma que PHP de la forma inicial que lo tenía desarrollado no era capaz de procesarlo aunque si funcionara en Postman, he tenido que obtener dicha información e iterar para que pueda ser procesada haciendo una adaptación para que funcione en Postman y Angular, en mi caso lo he tenido que adaptar para las peticiones POST y PUT, ya que con esas me pasaban todo tipo de parámetros por el body indiscriminadamente en formato JSON, los distinguía en el JSON, pero no lo ponía en el formato en el cual fue la API inicialmente concebida, lo cual causaban errores.

El proyecto ha sido desarrollado siguiendo un orden lógico y coherente siguiendo el siguiente orden durante el desarrollo:

- Base de datos MySQL.
- API RESTFUL con PHP y MySQL.
- Pruebas a la API.
- Maquetación web.

- Aplicación web con Angular y adaptaciones a la API para poder funcionar con Angular.

- Pruebas a Angular y a la API.

6. Implantación del Proyecto:

6.1 Plan de implantación

La implantación no lleva mucho tiempo y no es complicada si se sabe como, hay dos posibilidades de uso y dependiendo de estás se requiere de un tipo de instalación u otra.

- **Instalación en local para pruebas o con fines de desarrollo**: Esta precisa de la instalación de las dependencias necesarias (PHP, MySQL, Apache/Nginx, Node.js y Angular CLI).

- **Instalación para la puesta en producción**: Esta precisa de un servidor web que sea capaz de ejecutarlo (PHP, MySQL, Apache/Nginx, Node.js y crear un build del proyecto creado en Angular). **Se recomienda encarecidamente cambiar las claves por defecto por temas de seguridad**.

6.2 Manual de instalación

6.2.1 Instalación en local para pruebas o desarrollo

6.2.1.1 Instalación de las dependencias para el entorno local desde Windows

En mi caso se ha utilizado el WampServer el cual es un launcher de un servidor web con Apache, PHP y base de datos MySQL o MariaDB, para cumplir con las dependencias de la API, en este caso se ha utilizado MySQL aunque también puede valer el otro.

Web de descarga de WampServer con posibilidad de descargar módulos y actualizaciones y las distintas versiones: https://wampserver.aviatechno.net/

En este caso se está usando la versión 3.2.5 – 64bit con los siguientes módulos instalados y descargados de las dependencias que precisa el proyecto (WampSever ya las incluye, pero las cambie debido a que yo trabajo con esas versiones):

- Apache 2.4.48
- PHP 7.4.23
- MySQL 5.7.31

La instalación de WampServer es muy sencilla, nos pedirá la ruta de instalación, en mi caso pongo la que viene por defecto "C:\wamp64", en este manual se mostrará la instalación con la versión 3.2.5.

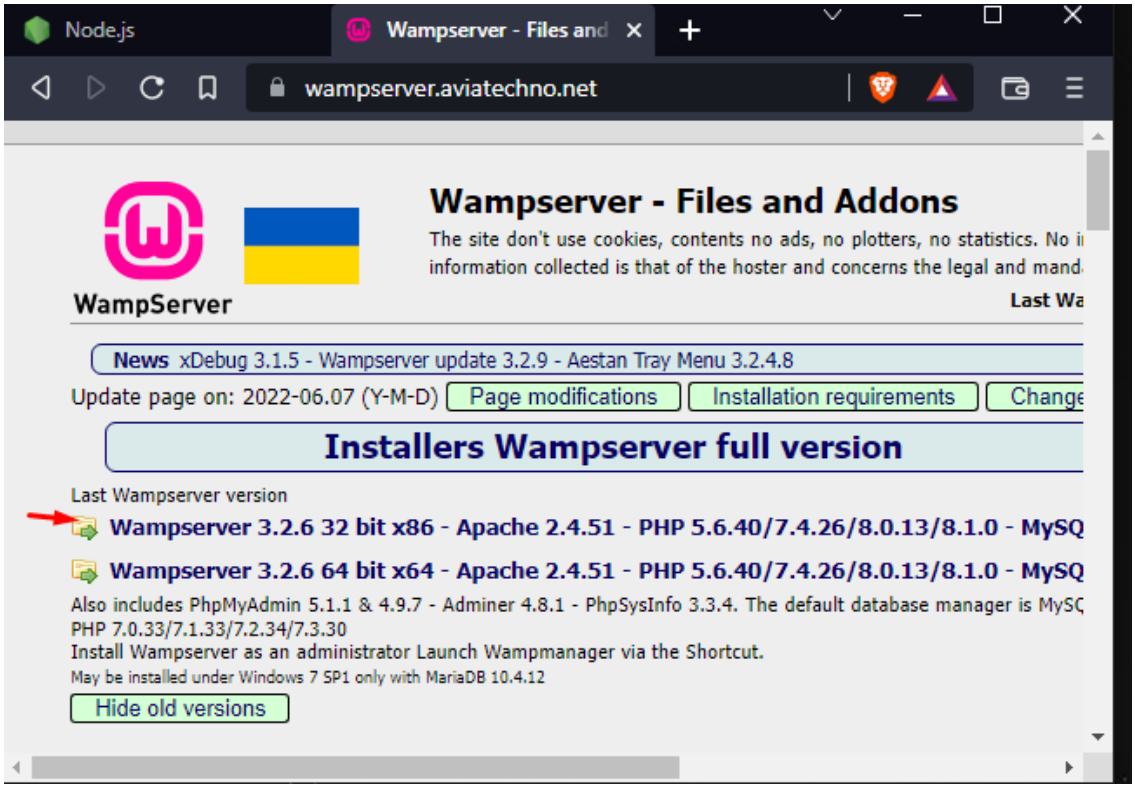


Imagen 6.2.1.1.1: Página de descarga de WampServer
https://wampserver.aviatechno.net/

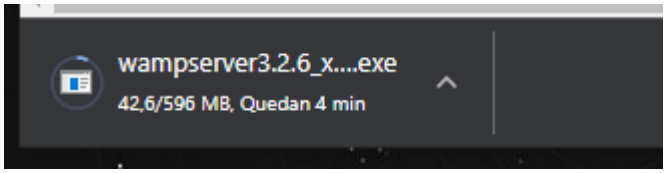


Imagen 6.2.1.1.2: Descargando WampServer

Imagen 6.2.1.1.3:
Ejecutando WampServer

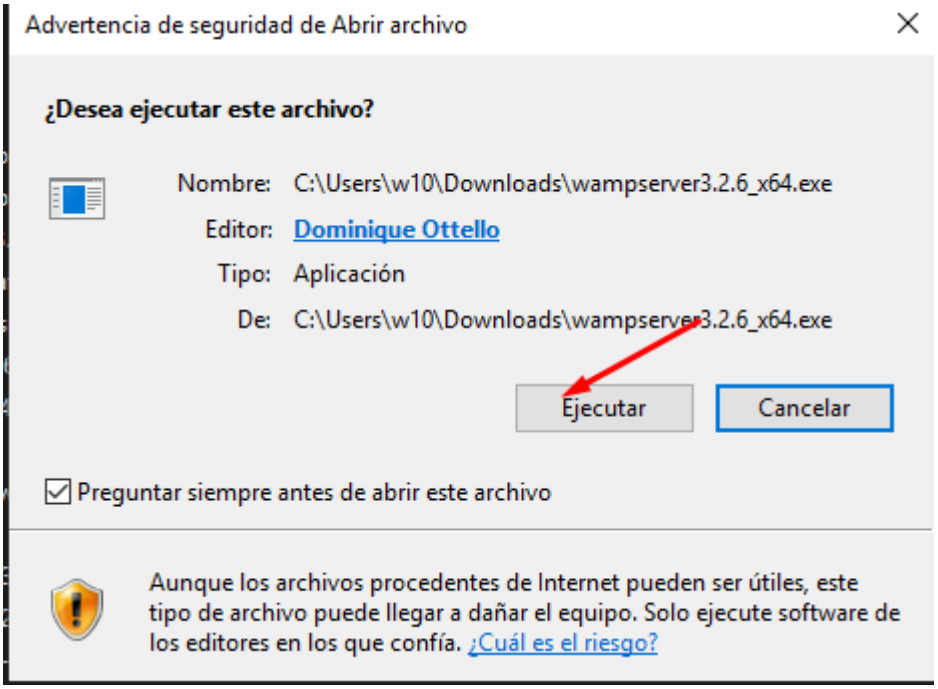


Imagen 6.2.1.1.4: Ejecutado WampServer

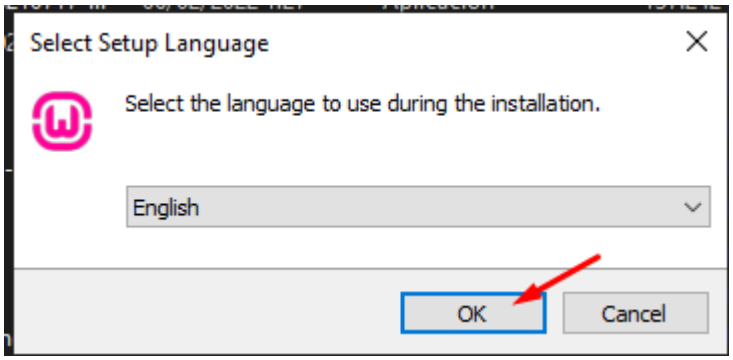


Imagen 6.2.1.1.5: Instalando WampServer

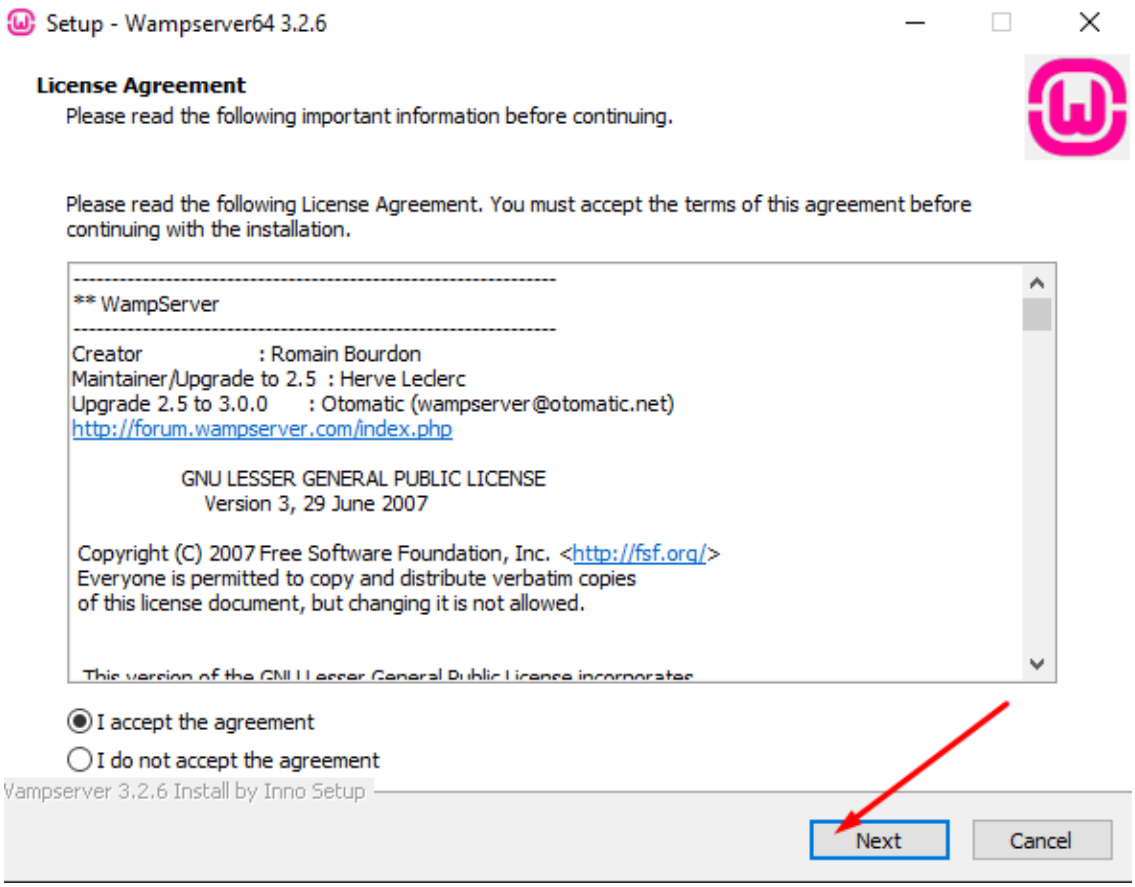


Imagen 6.2.1.1.6: Condiciones y términos de uso de WampServer

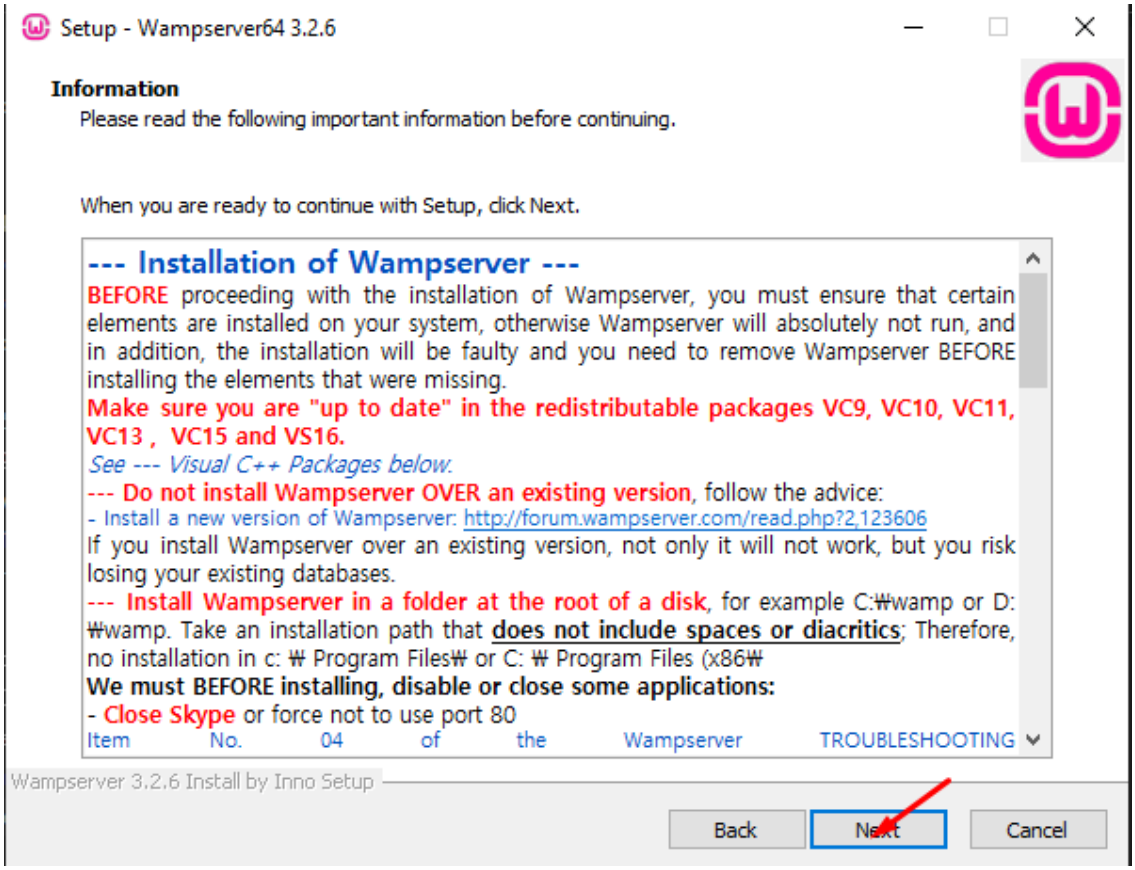


Imagen 6.2.1.1.7: Instalación de WampServer

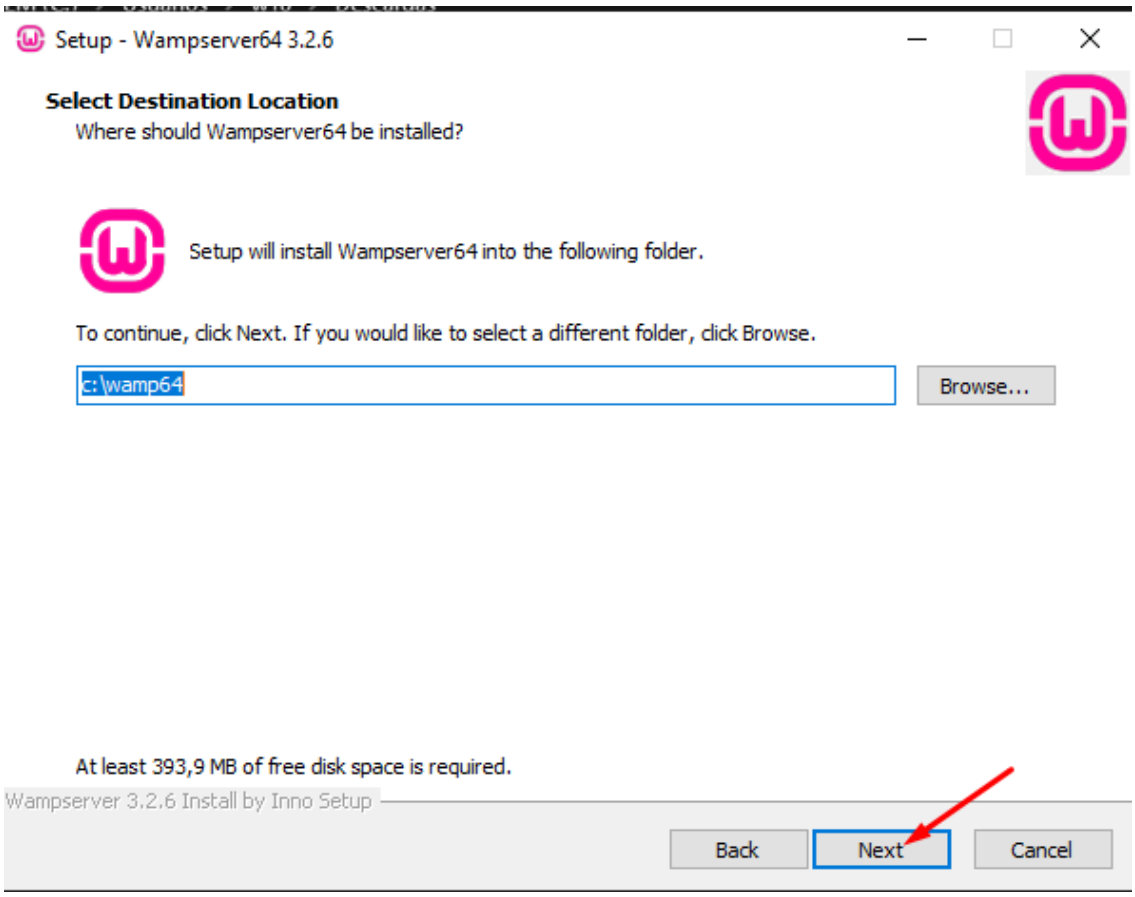


Imagen 6.2.1.1.8: Selección del directorio de instalación de WampServer

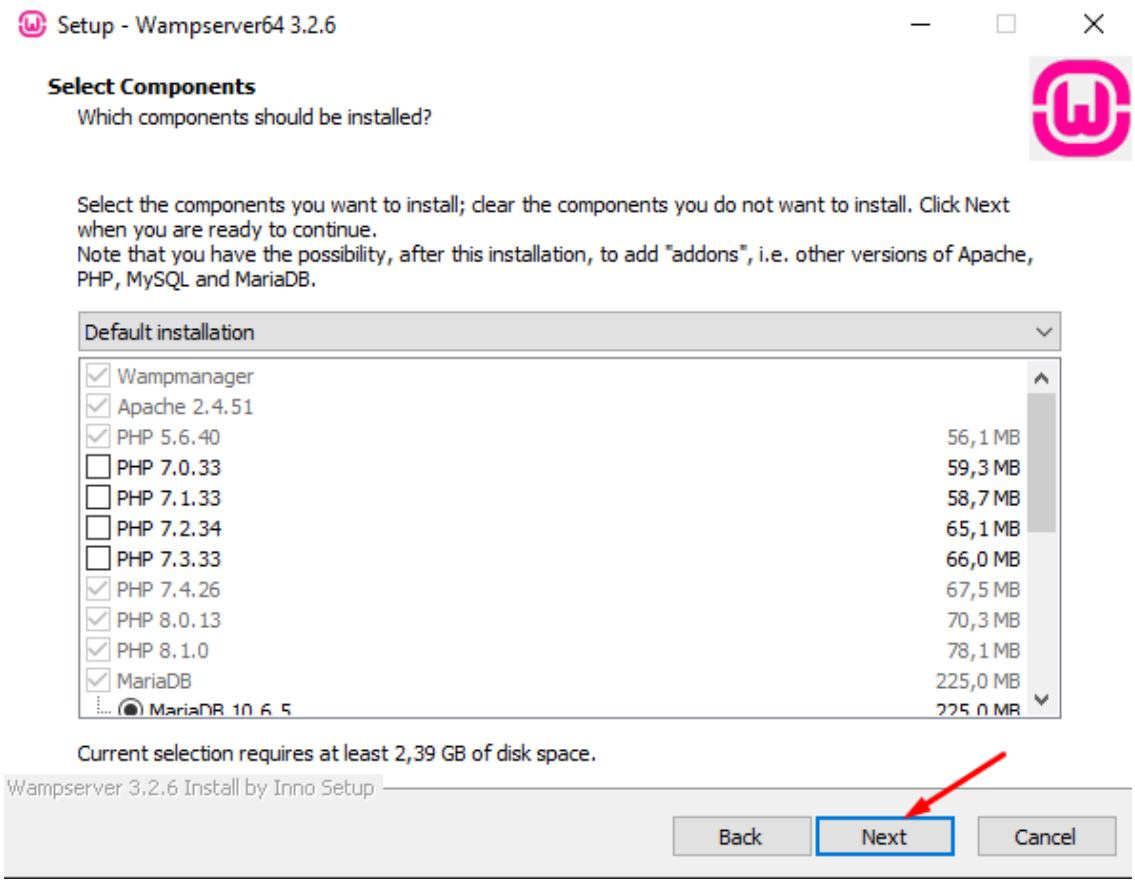


Imagen 6.2.1.1.9: Selección de paquetes a instalar en la instalación de WampServer

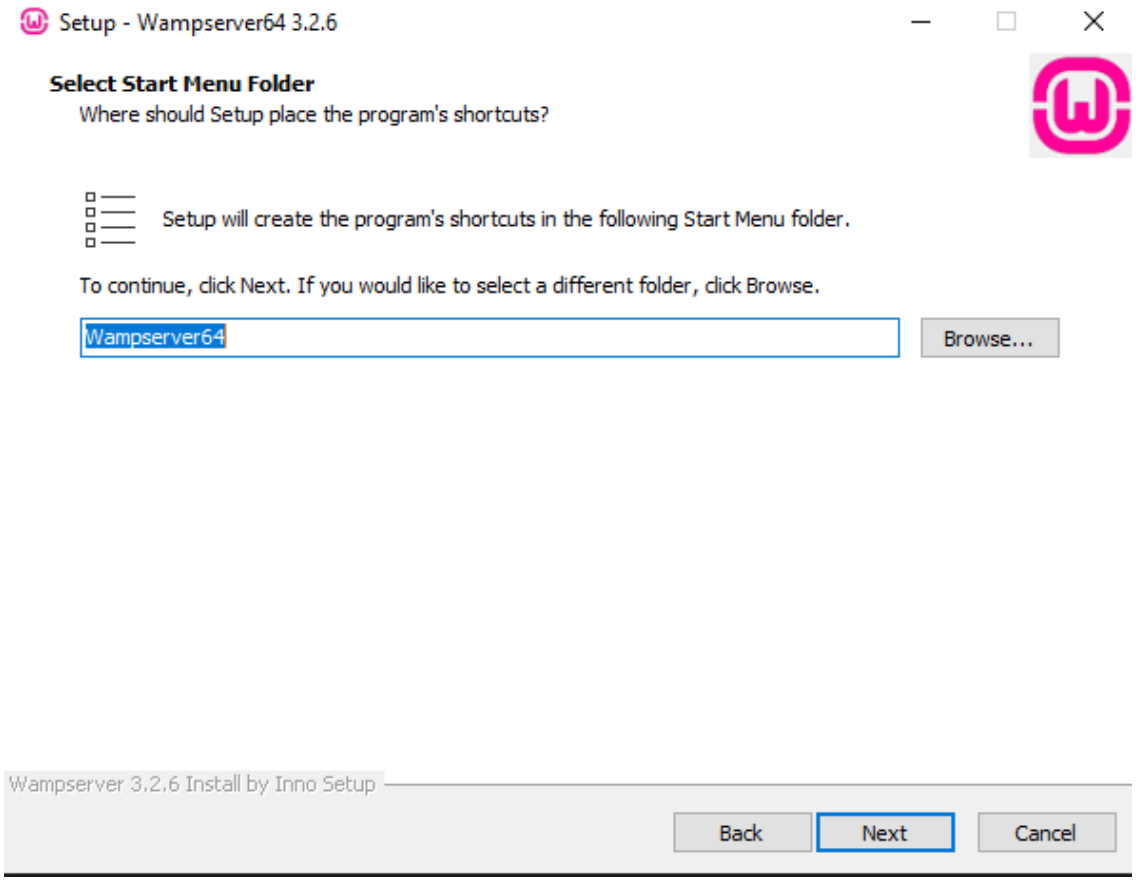


Imagen 6.2.1.1.10: Determinar el nombre de WampServer en el menú de inicio

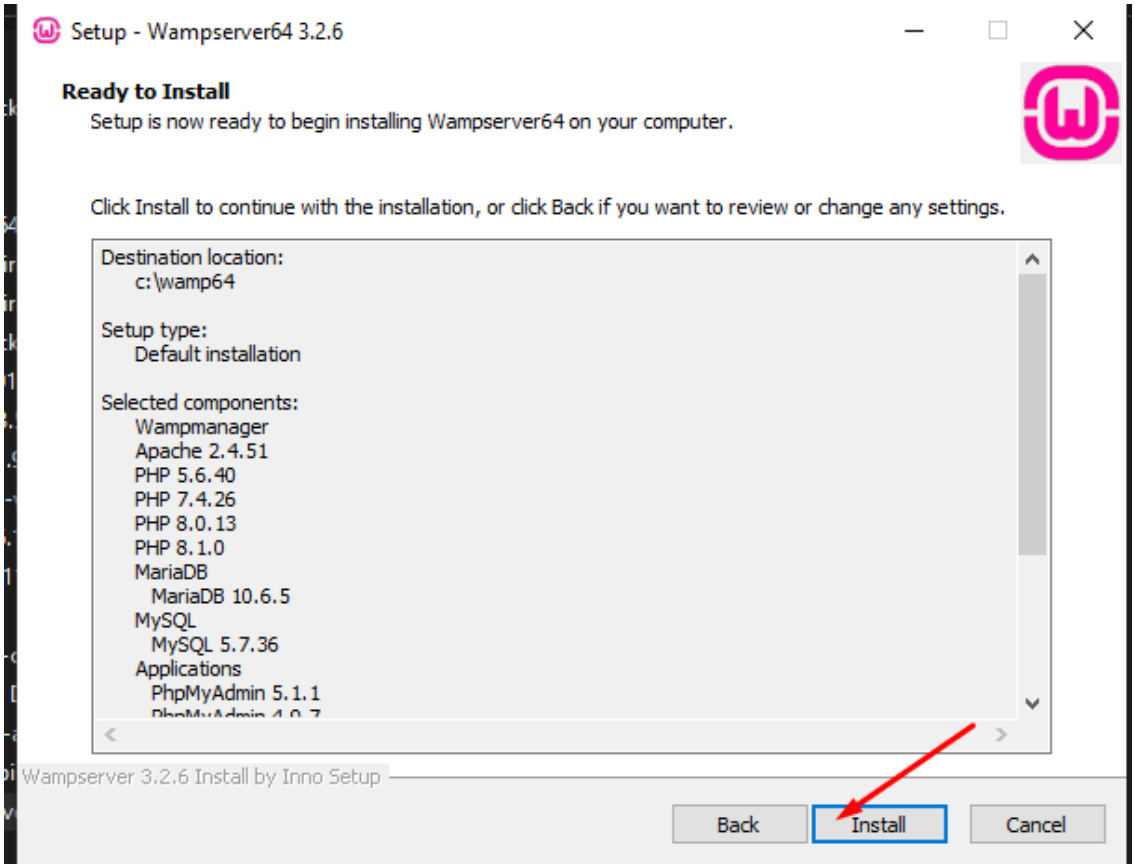


Imagen 6.2.1.1.11: Instalación de WampServer

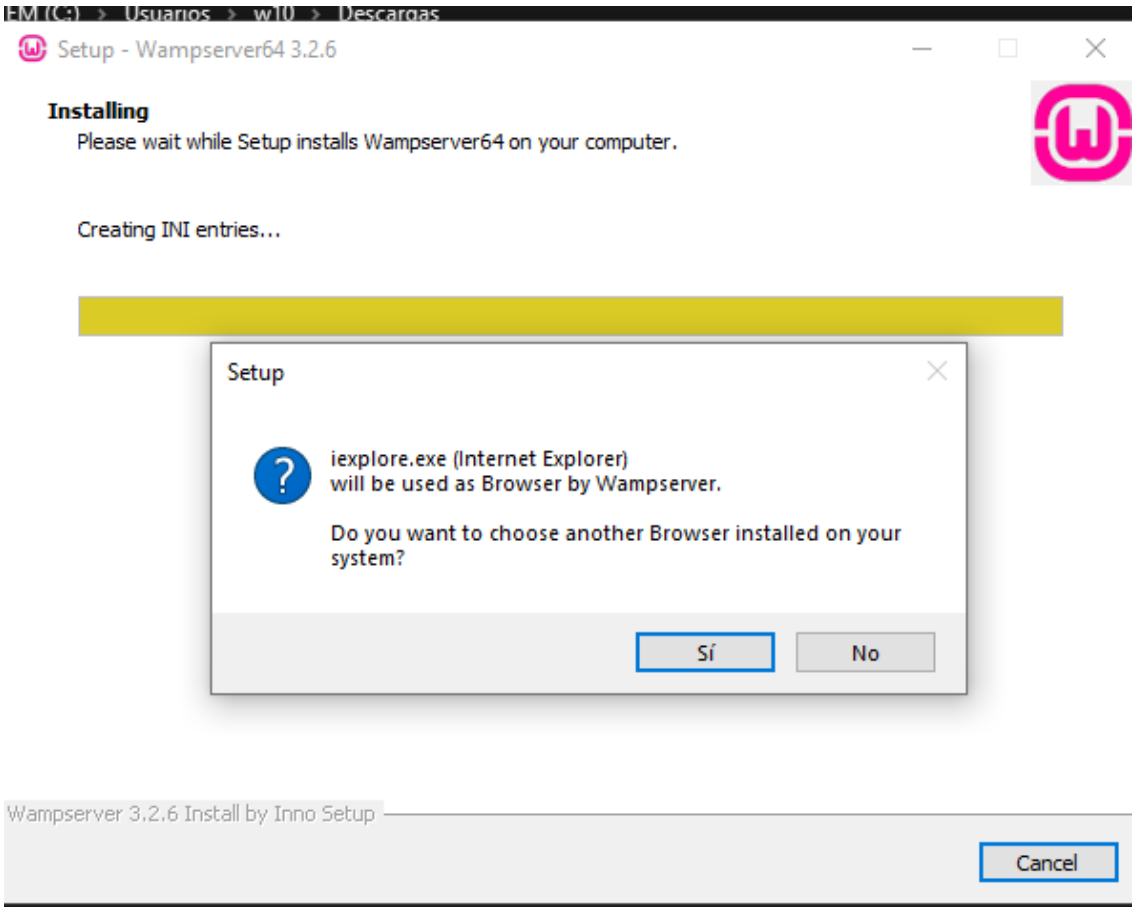


Imagen 6.2.1.1.12: Selección del navegador predeterminado para WampServer

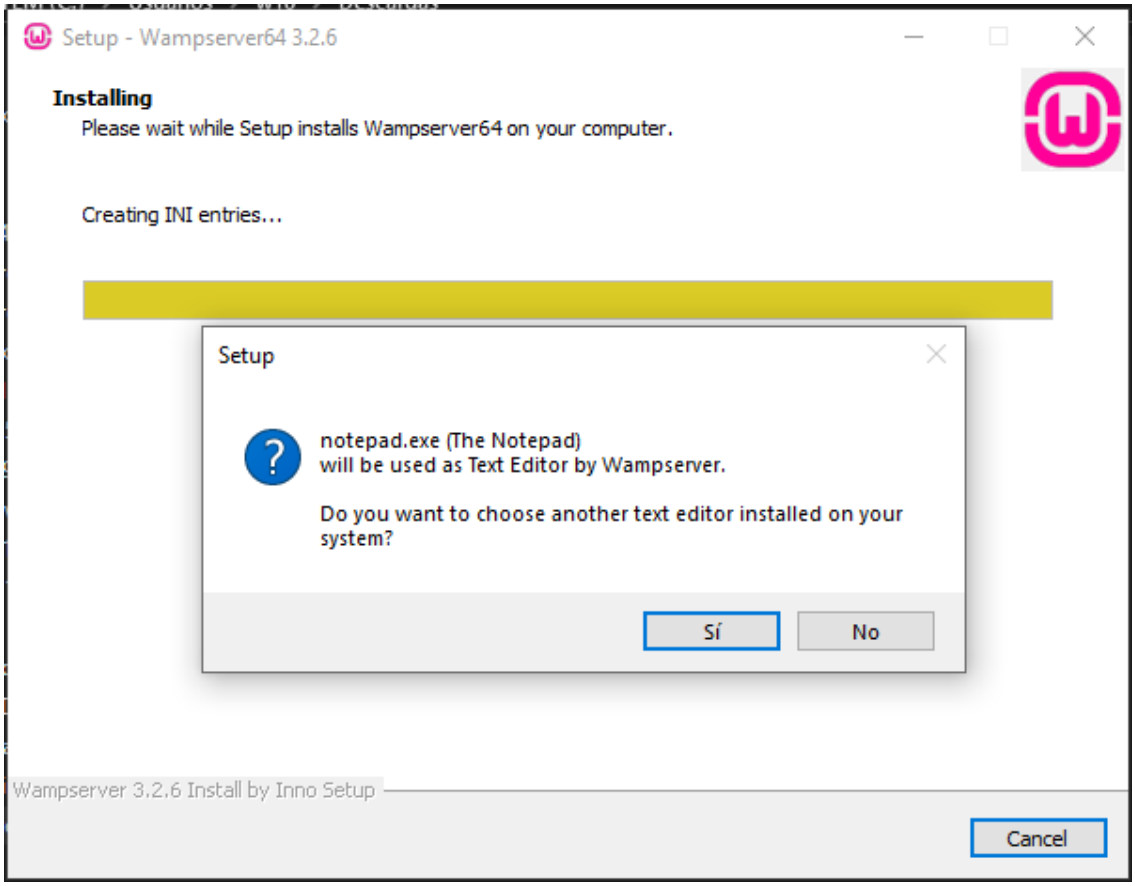


Imagen 6.2.1.1.13: Selección del editor de texto plano para WampServer

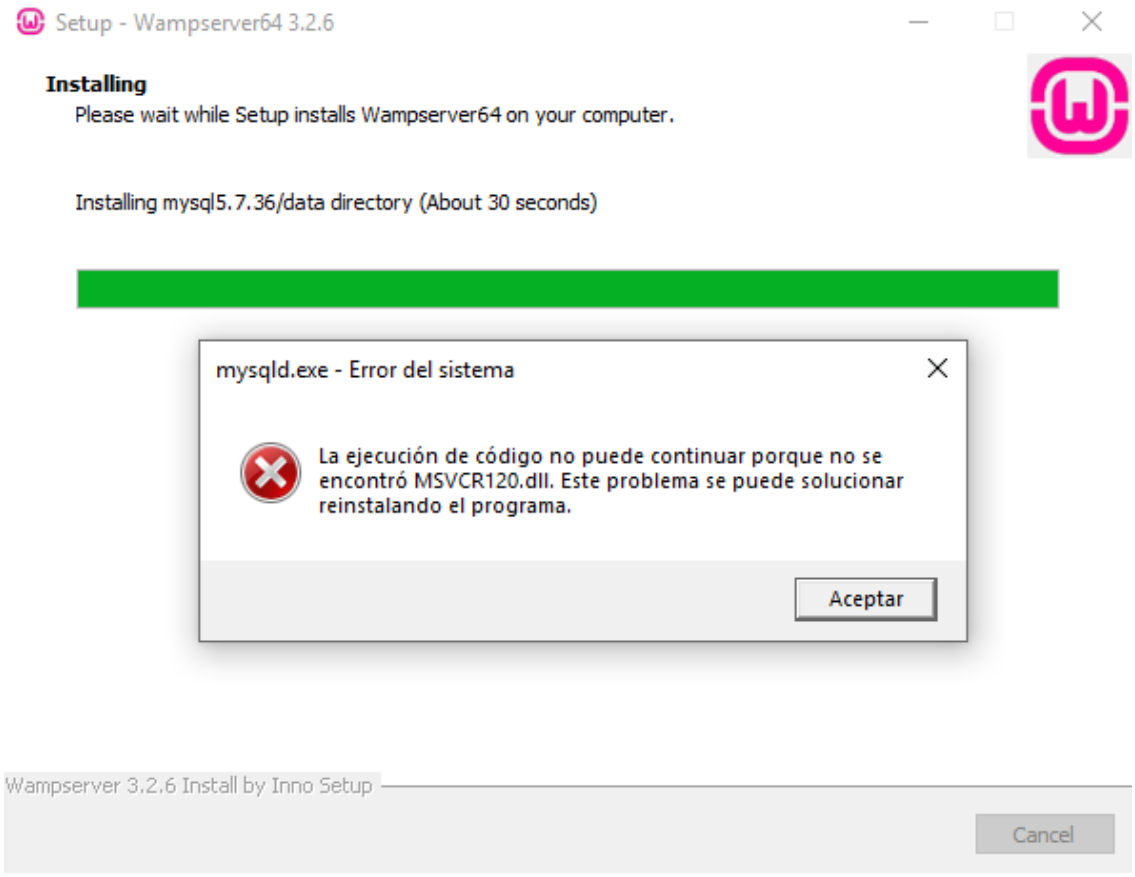


Imagen 6.2.1.1.14: Error en la instalación que podría ocurrir

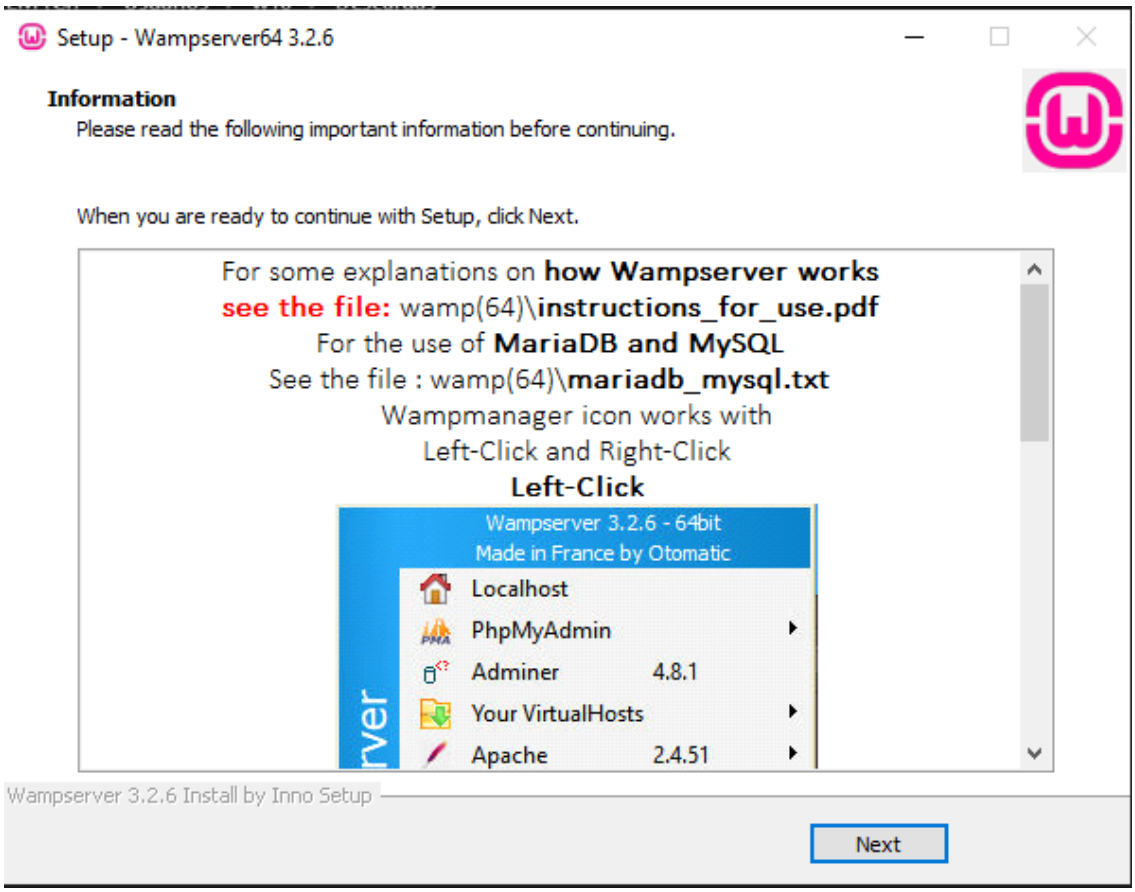
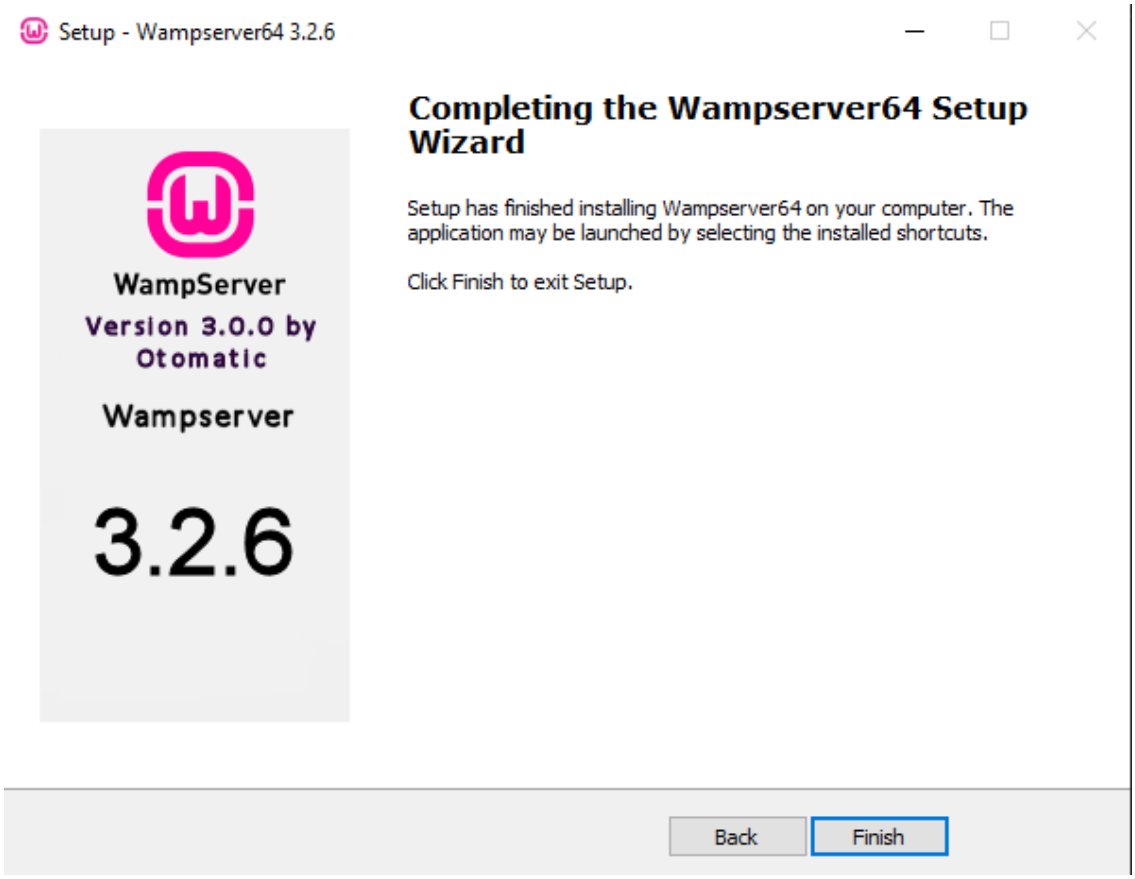


Imagen 6.2.1.1.15: Instalación de WampServer



Instalación 6.2.1.1.16: Finalización de la instalación de WampServer

Solución a la falta de dependencias para ejecutar WampServer, se soluciona instalado los siguientes paquetes de actualización, los cuales son los paquetes de Microsoft Visual C++:

- https://www.microsoft.com/en-us/download/details.aspx?id=30679
- https://www.microsoft.com/en-us/download/details.aspx?id=40784
- https://www.microsoft.com/en-us/download/details.aspx?id=48145
- https://www.microsoft.com/en-us/download/details.aspx?id=53587

En caso de no funcionar los enlaces podrás buscarlos en el Catálogo de descargas de Microsoft Update https://www.catalog.update.microsoft.com/Home.aspx, otra posibilidad es que al actualizar Windows todas esas dependencias se instalen y se pongan actualizadas.

Una vez actualizado y con las dependencias instaladas, instala WampServer nuevamente desinstalando antes.

En caso de no funcionar deberás de instalar las rutinas de Microsoft, es posible que te la instale Windows Update o de lo contrario obtenerlo del catalogo de Microsoft, otra alternativa es utilizando el instalador de aplicaciones Ninite https://ninite.com/, el cual a mi me funciona bastante bien.

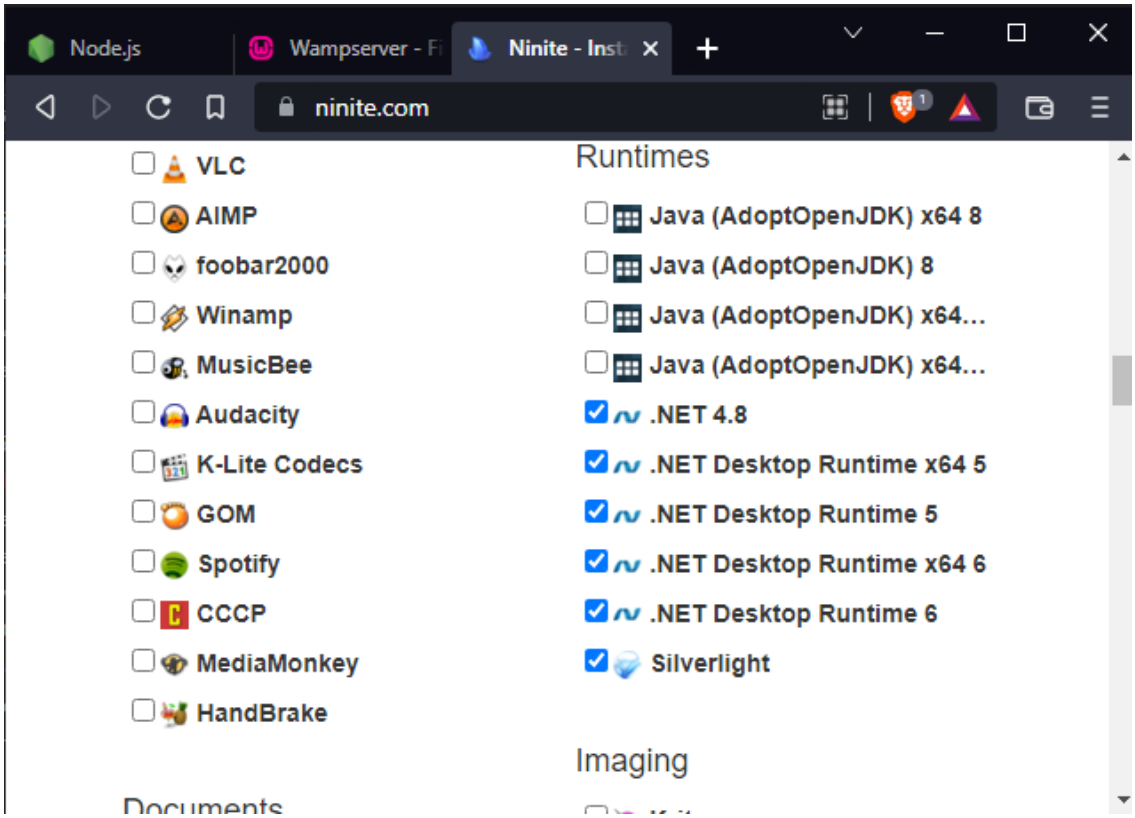


Imagen 6.2.1.1.17: Web del instalador Ninite https://ninite.com/

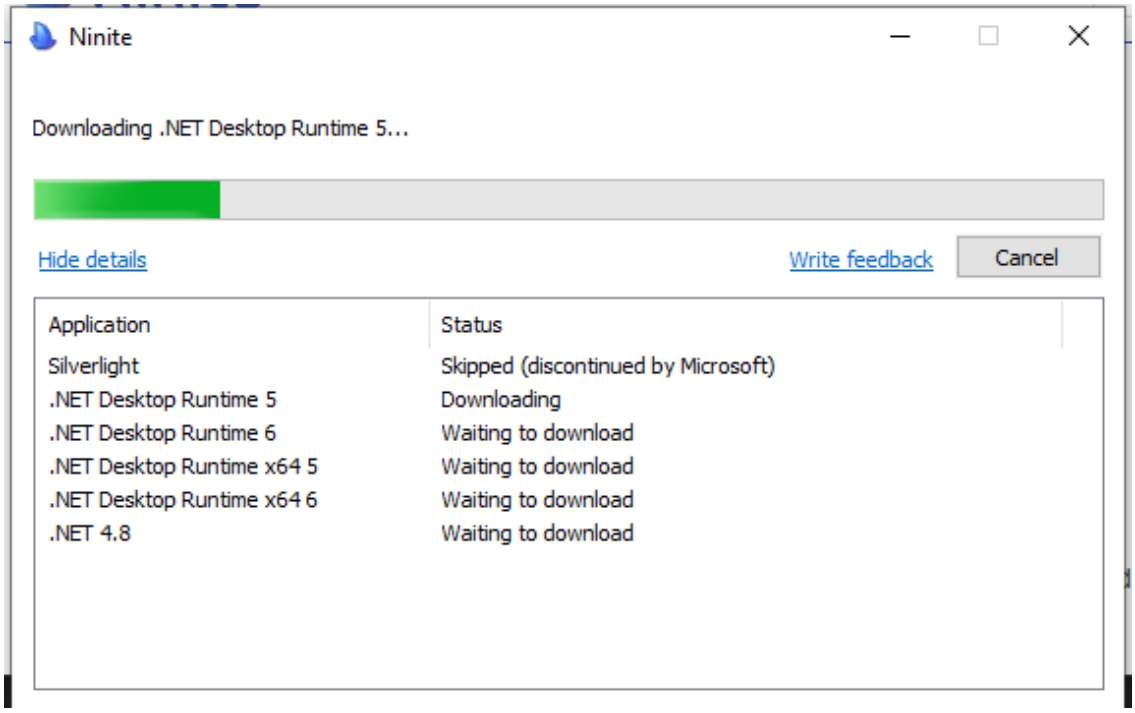


Imagen 6.2.1.1.18: Instalación de las rutinas de Microsoft desde el instalador Ninite

Reiniciamos y volvemos a desinstalar y instalar WampServer y nos aseguramos de que Windows está actualizado.

En caso de que alguno de los servicios no se inicie es posible que sea a causa de que ese puerto está ocupado, WampServer nos brinda herramientas para comprobar si el puerto está ocupado y cambiar dicho puerto, además de averiguar que proceso está ocupando dicho puerto.

Para acceder a las herramientas que nos ayudan con el tema relativo a puertos en WampServer se encuentran al hacer clic derecho sobre el icono de WampServer en la barra de tareas.

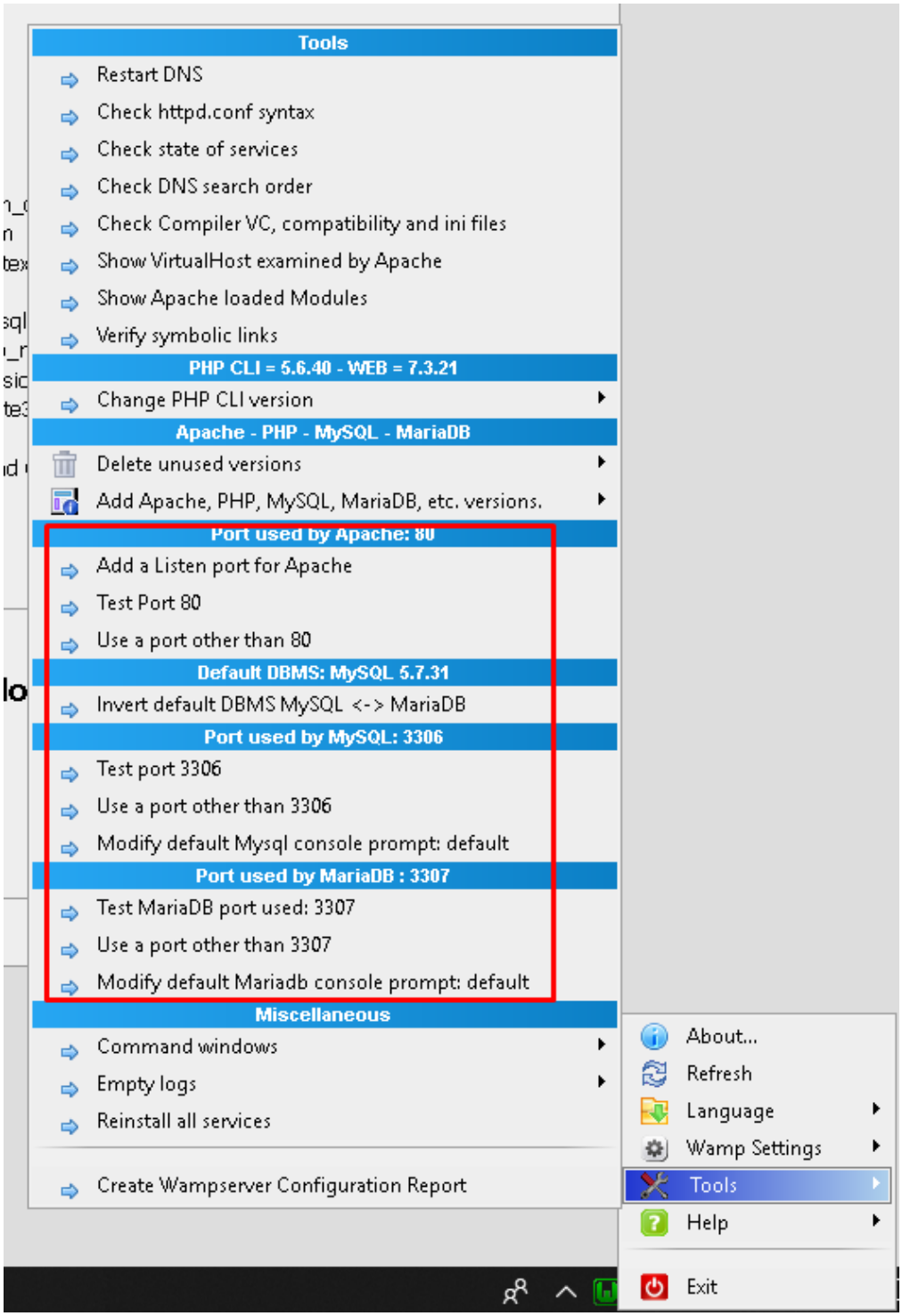
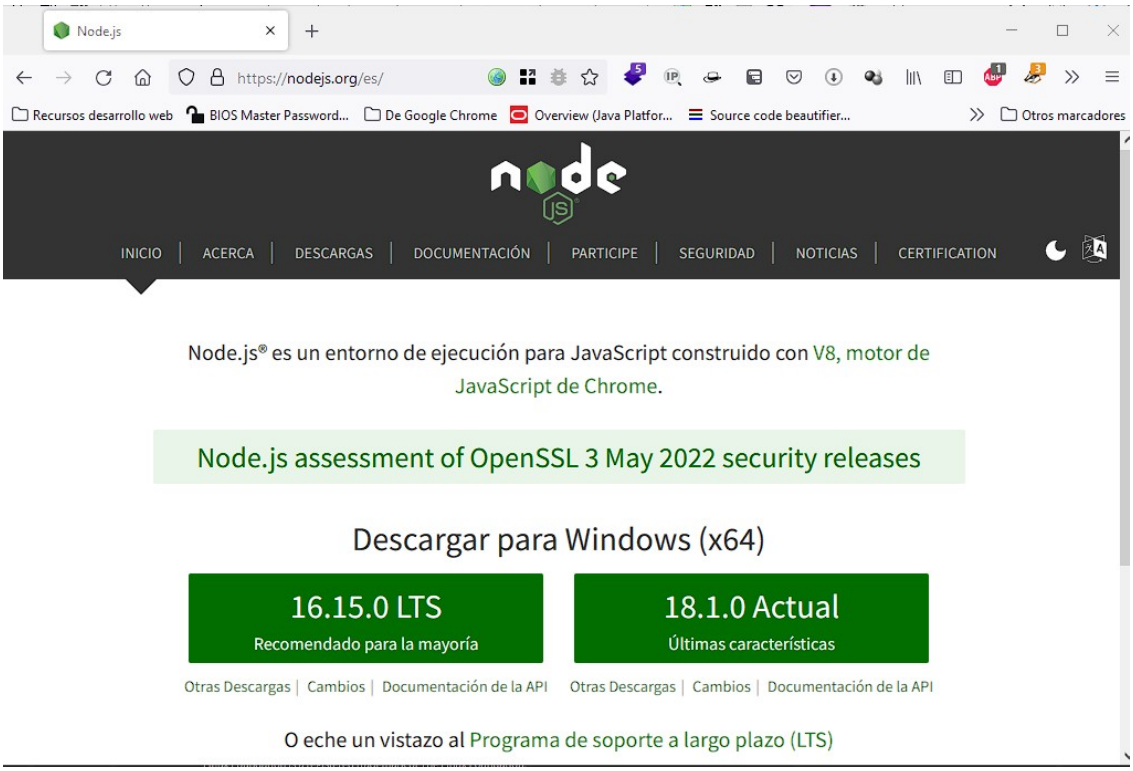


Imagen 6.2.1.1.19: Herramientas de WampServer

6.2.1.2 Instalación de Angular CLI para el entorno local

El primer paso a realizar para utilizar cualquier framework de JavaScript es instalar/actualizar Node.js, en mi caso y lo recomendable es utilizar la ultima versión estable (LTS). https://nodejs.org/es/

En mi caso que lo estoy usando en Windows hay que descargarlo de la web y es el típico instalador de Siguiente … Siguiente a golpe de clic.



6.2.1.2.1: Web de Node.js https://nodejs.org/es/

Una vez instalado hay que ir a la consola de comandos y actualizar el **npm** (Que es el gestor de paquetes de node para bajarnos las dependencias más actuales del framework).

```
npm install -g npm@latest
```

La opción -g es para la instalación global, sin ella solo sería a nivel local (por ejemplo, librerías y dependencias utilizadas en un proyecto de Angular)

Borramos la cache para evitar problemas, de lo contrario es posible que utilice los paquetes de la cache en lugar de los que hemos descargado.

```
npm cache clean --force
```

Una vez hecho esto deberíamos de desactivar las auditorias de npm para evitar fallos durante la instalación, además que cuando se tratan de frameworks estos ya han sido testeados y vienen seguros.

```
npm set audit false
```

Ahora toca ejecutar un par de comandos para el caso de que tengas instaladas versiones antiguas de **Angular CLI** (Angular CLI es el programa que nos permite generar e instalar proyectos de Angular, es el más importante para trabajar con Angular, es el que nos generará los ficheros y nos dará la funcionalidad de Angular entre muchas otras funcionalidades).

En caso de tener versiones antiguas de Angular hay que desinstalarlos.

```
npm unistall Angular -g angular-cli
```

```
npm unistall -g @angular/cli
```

Limpiando la cache de datos antiguos.

```
npm cache clean --force
```

Y entonces ya podremos instalar la ultima versión estable de Angular.

```
npm install -g @angular/cli@latest
```

Una vez instalado podemos comprobar que tenemos Angular instalado.

```
ng --version
```

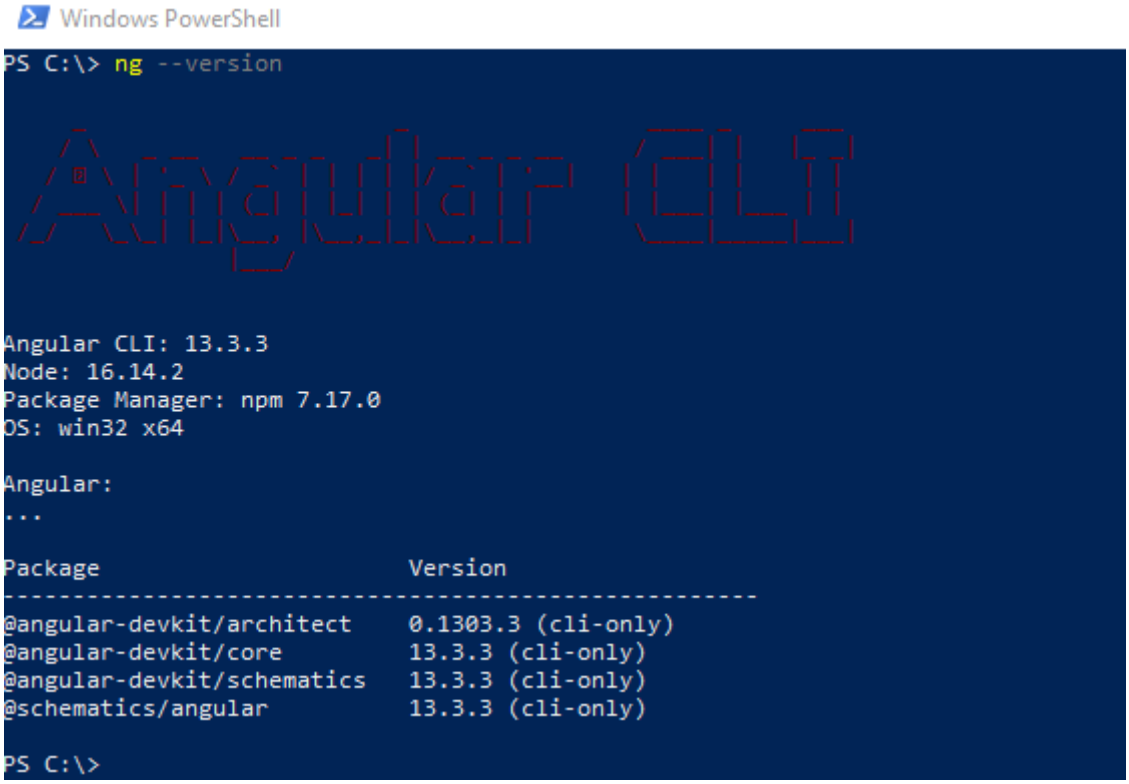


Imagen 6.2.1.2.2: Comprobando la versión instalada de Angular CLI

Al tratar de ejecutar comandos de Angular es posible que nos arroje un error de PowerShell, si lo intentamos ejecutar desde PowerShell (desde CMD no pasaría), **debido a que por defecto se tiene restringido en PowerShell la ejecución de de scripts**.

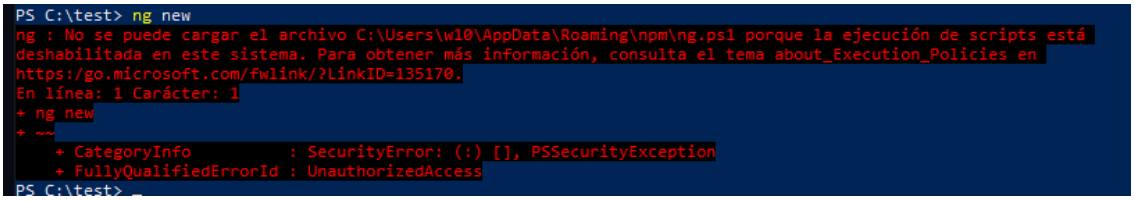


Imagen 6.2.1.2.3: Error de PowerShell en caso de tener restringido la ejecución de scripts (que de hecho es el valor por defecto)

Para solventar esto hay que **habilitar la ejecución de scripts en PowerShell** abrimos una nueva terminal con privilegios de administrador y cambiamos la política de ejecución de scripts a cualquiera sin firmar (esto con lleva cierto riesgo pero en este caso es necesario configurar con esa directivas).

Set-ExecutionPolicy Allsigned

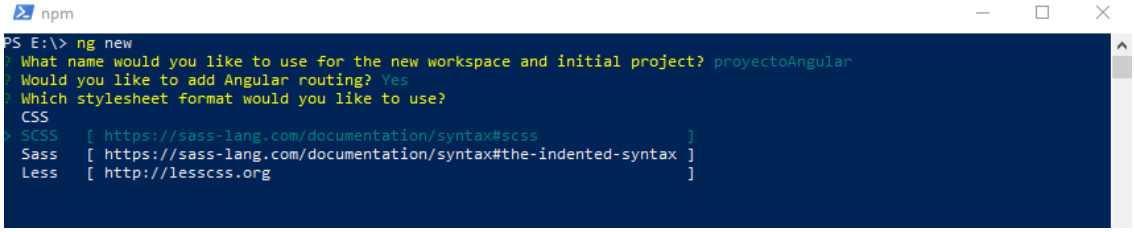


Imagen 6.2.1.2.4: Cambiando la política de ejecución de scripts en PowerShell

6.2.1.3 Instalación de Postman y importación de una colección de datos

El primer paso pasa por descargar Postman de su web oficial https://www.postman.com/, una vez descargado la instalación basta con las opciones por defecto.

Una vez instalado abrimos Postman **importamos el fichero en la carpeta del proyecto llamado "APIREST FULL Proyecto final DAW Juan Molina 2022.postman_collection.json".**

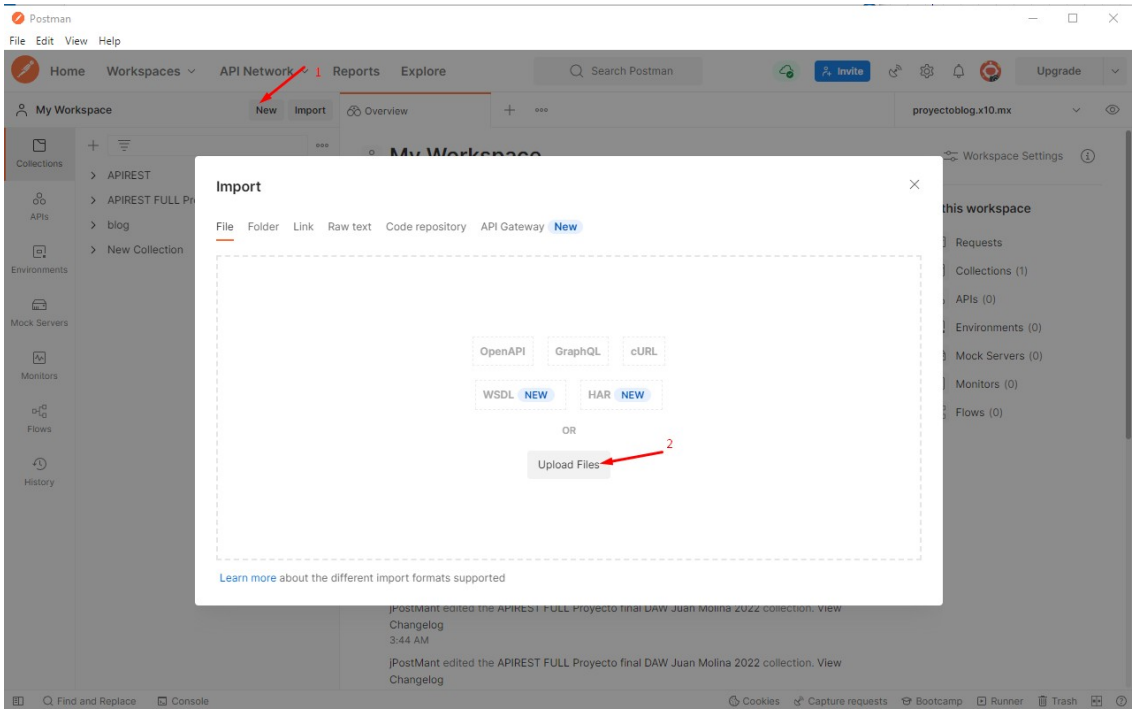


Imagen 6.2.1.3.1: Importación de una colección de datos a Postman

6.2.1.4 Despliegue de la aplicación en local

Copiamos la carpeta del proyecto en la carpeta publica del servidor WampServer la cual se encuentra en la carpeta de instalación y se llama **www**.

Una vez copiada ejecutamos WampServer y importamos la base de datos, tecleamos localhost en la barra de direcciones de cualquier navegador web y luego hacemos clic en PhpMyadmin (En caso de no aparecer acceder por la dirección http://localhost/phpmyadmin/ → Accedemos con nuestras credenciales (En este caso usuario root sin contraseña) → Import → Seleccionamos un archivo que hay dentro de la carpeta del proyecto llamada blog.sql

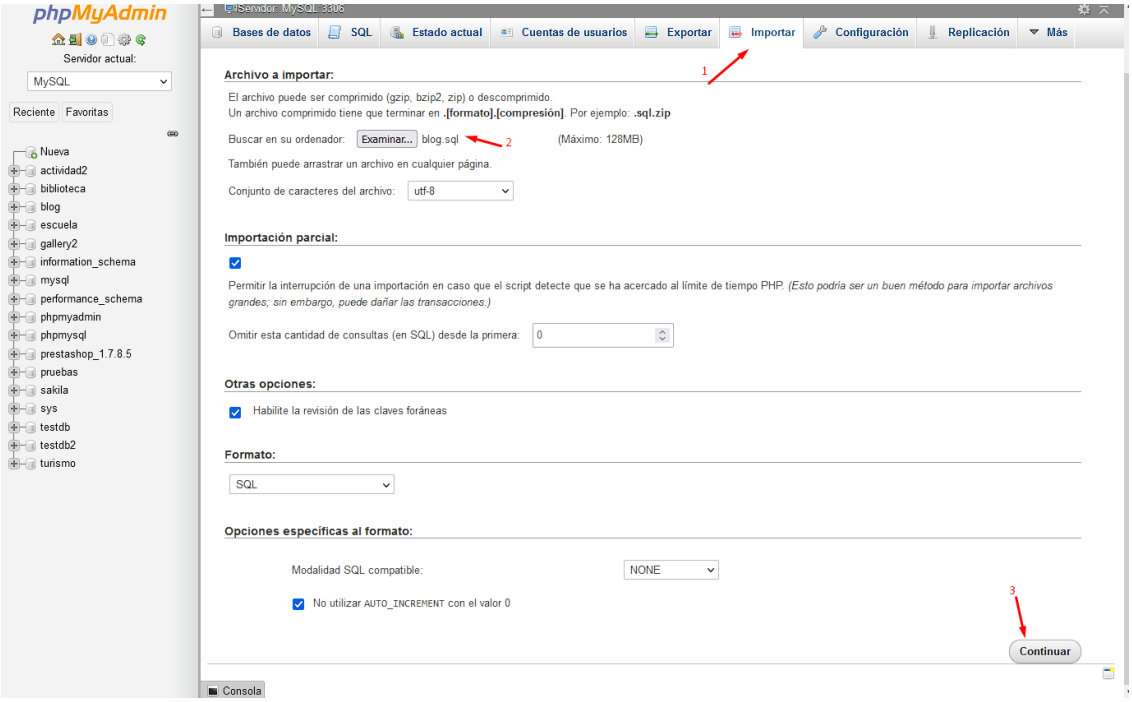


Imagen 6.2.1.4.1: Importación de la base de datos en phpMyAdmin

Por último abrimos una consola de Windows (ya sea PowerShell o CMD), nos situamos en la ruta donde se encuentra el proyecto hecho con Angular y ponemos en marcha el proyecto de Angular.

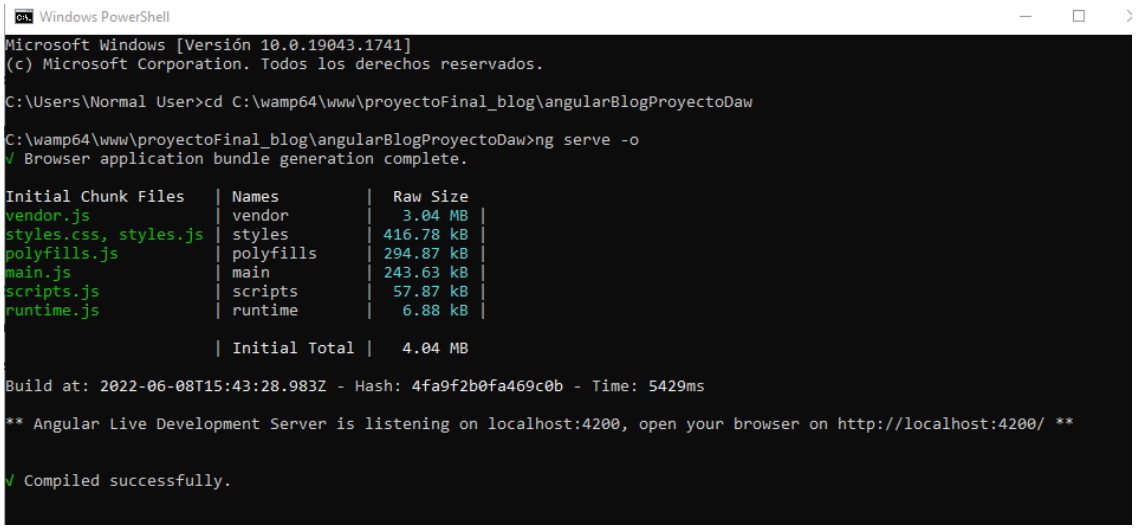


Imagen 6.2.1.4.2: Ejecutando el watch de Angular

Para ejecutar Angular en un entorno de desarrollo / local hay que utilizar el comando **"ng serve"**, como opción opcional puedes usar la opción "-o" para que se te abra automáticamente en el navegador, de lo contrario tendrás que copiar y pegar la dirección en la barra de direcciones tu mismo, el puerto por defecto es el 4200, pero si ejecutas una nueva instancia mientras tienes otro proyecto ejecutándose, **Angular le asigna un puerto aleatorio distinto a 4200 y que este libre**.

6.2.2 Configuraciones y ajustes en el proyecto muy importantes a tener en cuenta

Antes de ponerlo en producción es muy importante tener en cuenta las siguientes configuraciones dentro del código:

- En la API con PHP:
  - **Tener siempre en cuenta los endpoints,** dependiendo de si está en la carpeta raíz o en otra ruta.

    Routes → routes.php

    - Hay que editar el primer bloque if y el tercero, "count($routesArray) == **2" el número que le sigue el valor más bajo es 1 en caso de ser desde la raíz** y en caso de estar dentro de una carpeta en raíz por ejemplo sería el valor 2 como sale por defecto, todo depende de la ruta donde lo sitúes

la API y a partir de donde tiene que empezar a detectar parámetros la API.

- **Muy importante por temas de seguridad cambiar el API KEY y credenciales de la base de datos.**

  models → connection.php

  - Método **apikey**: Para el token de acceso.

  - Método **infoDatabase**: Para las credenciales y parámetros de conexión a la base de datos.

- **Muy importante cambiar la clave de excepciones por defecto por temas de seguridad.**

  El token de excepciones solo afecta para las peticiones PUT (actualización) **y las de POST** (guardado) **el cual te excluye de estar logueado**, el cual se especifica con el parámetro GET execpt, y por el parámetro token se pasa el código de excepción.

  routes →services→ put.php routes→ services→ post.php

  Para encontrar la clave por defecto a modificar hay que buscar el comentario “Peticiones PUT para usuarios no autorizados” y editar la linea 60 en la condición if donde dice “no”, hay que sustituir ese string por uno que contenga una clave de excepción.

- **Cambiar los parámetros por defecto de usuario**, este paso es muy importante en caso de ponerse en producción, a ausencia de una interfaz en Angular no impide el cambio de esta debido a que la API está preparada y tenemos un fichero de colección de datos de Postman con una colección de datos con una gran variedad de peticiones HTTP que cubren todo lo que puede hacer la API RESTFUL dinámica que puede hacer.

  Antes que nada, hay que tener en cuenta que para modificar la contraseña necesitamos el Hash de la contraseña, para ello se ha desarrollado un pequeño script escrito en PHP que es capaz de generar dicho

Hash utilizando el mismo algoritmo definido para la API, el fichero a ejecutar en un servidor web se llama **"generadorPassword.php"**.

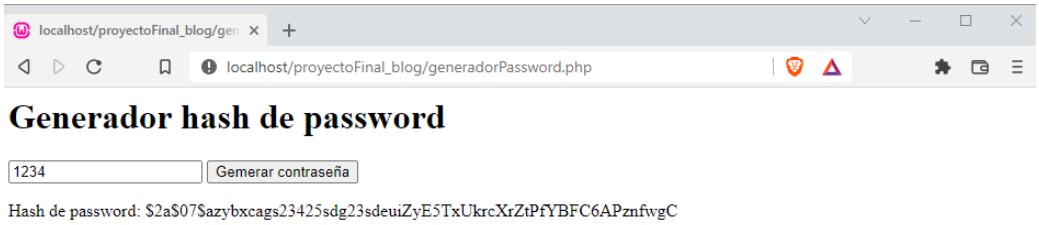


Imagen 6.2.2.1: Script generadorPassword.php

Vamos a Postman, entonces el primer paso imprescindible es loguearse.

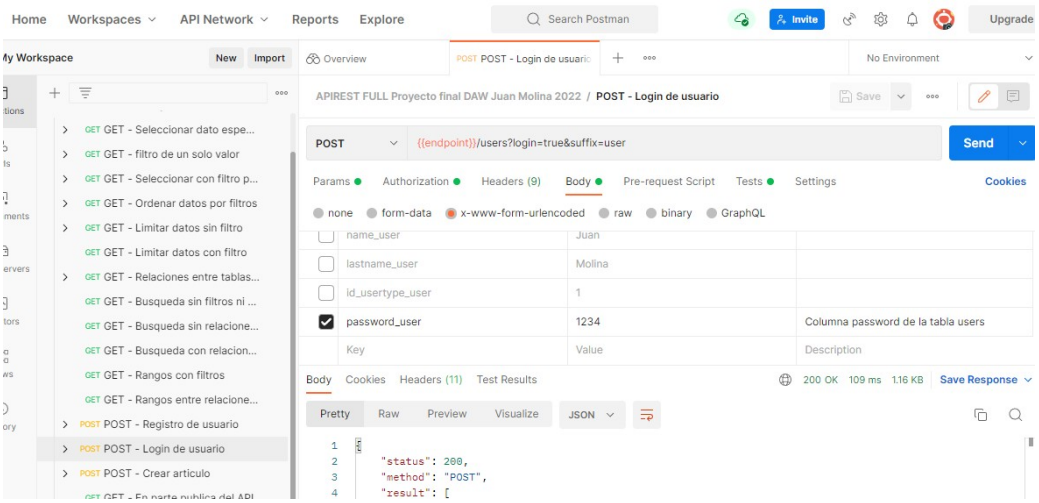


Imagen 6.2.2.2: Ejecutando método POST Login de usuario

Ahora entonces ejecutamos el método PUT – Actualizar registros, esto en caso de producción se debe de hacer para las dos cuentas existentes, la id 9 y la id 15.

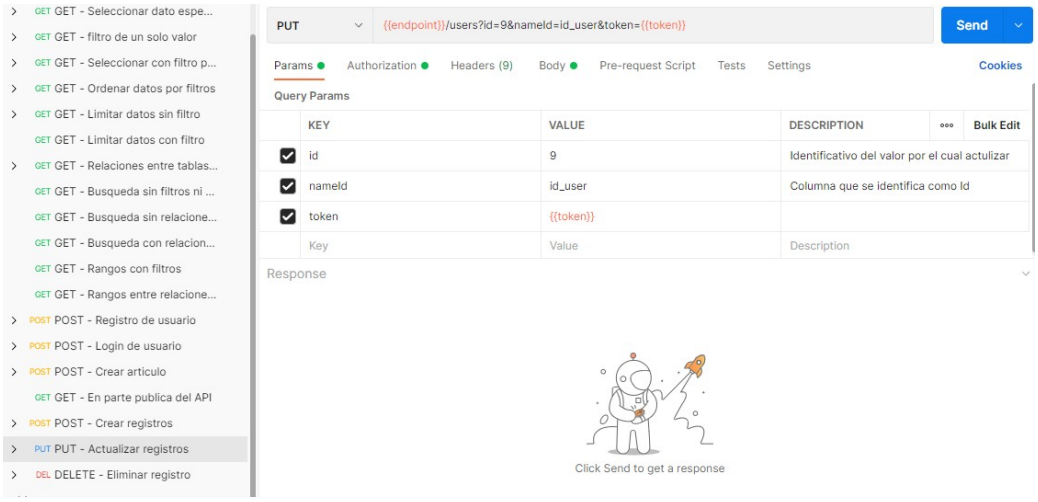


Imagen 6.2.2.3: Parámetros de la petición PUT para actualizar usuario

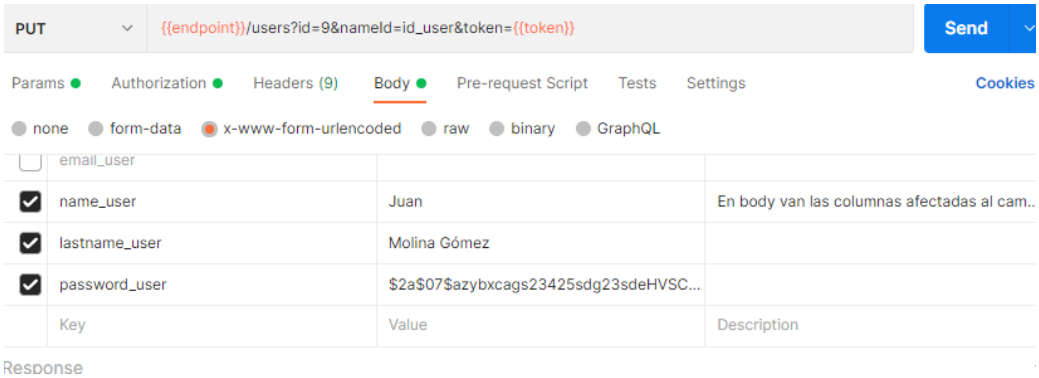


Imagen 6.2.2.4: Contenido del body para la petición PUT de la actualización de usuario

Una vez metido los parámetros y seleccionado en body lo que se quiere modificar le damos clic al botón que dice “Send” y si todo sale bien obtendremos una respuesta 200 indicando que todo ha salido bien.

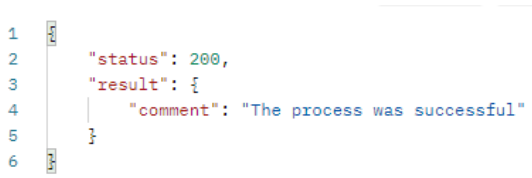


Imagen 6.2.2.5: Respuesta a la petición PUT de que ha salido bien

- En Angular:
    - En el servicio global.ts poner la URL de tu API y la API KEY que tiene por defecto.

      src → app → services → global.ts

Por último en Postman hay que tener en cuenta, que en el caso de ponerlo en producción tienes dos opciones, una es **modificar las variables de la colección de datos o más practico crear un Enviroment** (Entorno) en Postman, en el caso de crear un entorno tendrás que crear dichas variables poniendo los valores modificados que vas a usar en la aplicación en producción, de este modo por un lado sin modificar tienes las variables de la colección de datos para local y los entornos para otros servidores o ya puesto en producción.

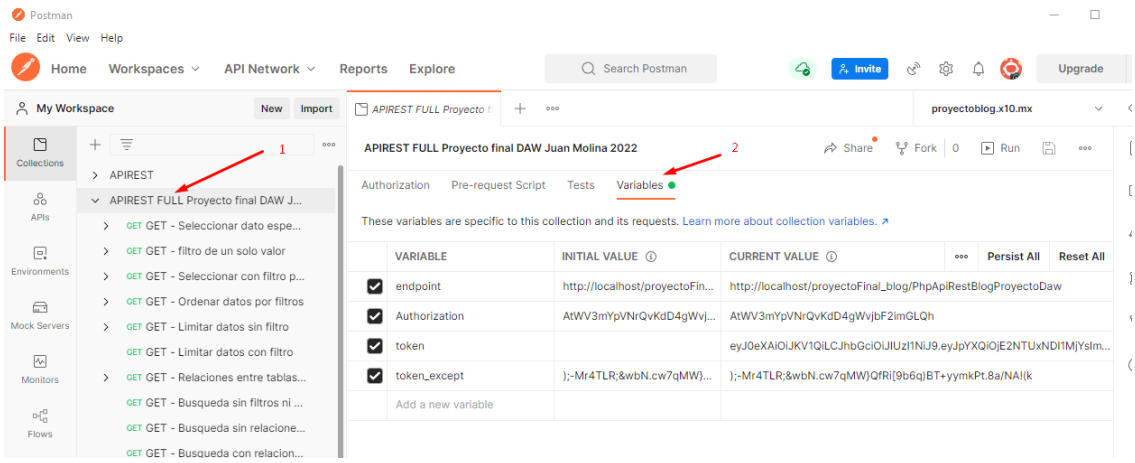


Imagen 6.2.2.6: Como acceder a las variables de una colección de datos en Postman

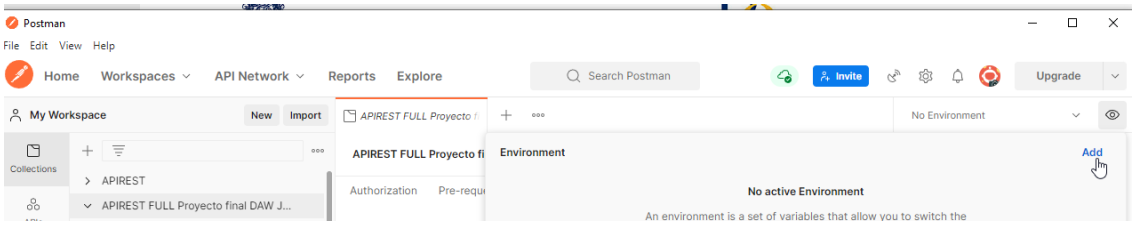


Imagen 6.2.2.7: Como añadir un nuevo entorno en Postman paso 1

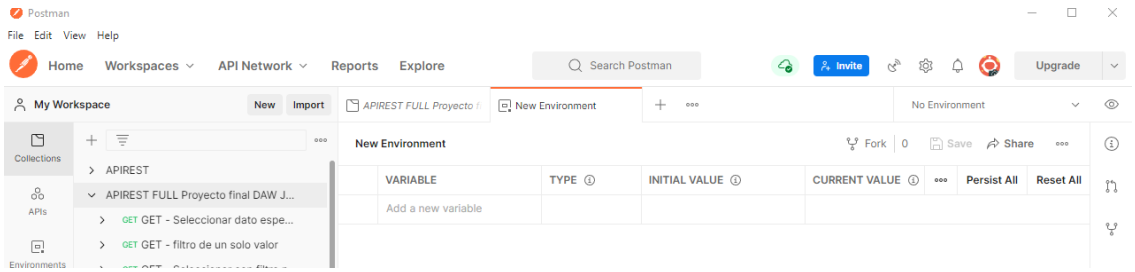


Imagen 6.2.2.8: Como crear un nuevo entorno en Postman paso final, aquí se pueden poner las variables y cambiar el nombre

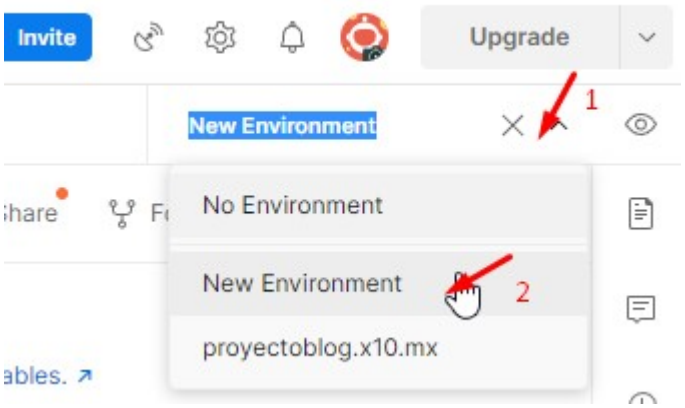


Imagen 6.2.2.9: Como editar un entorno en Postman paso 1

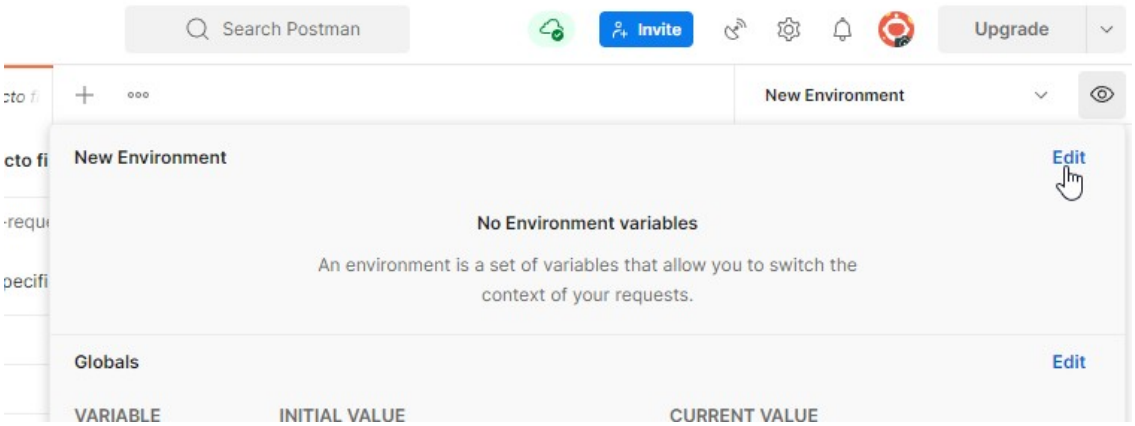


Imagen 6.2.2.10: Como editar un entorno de Postman paso 2 y por último nos encontramos lo mismo que la imagen 6.2.2.8 donde se pueden crear variables y editar sus valores

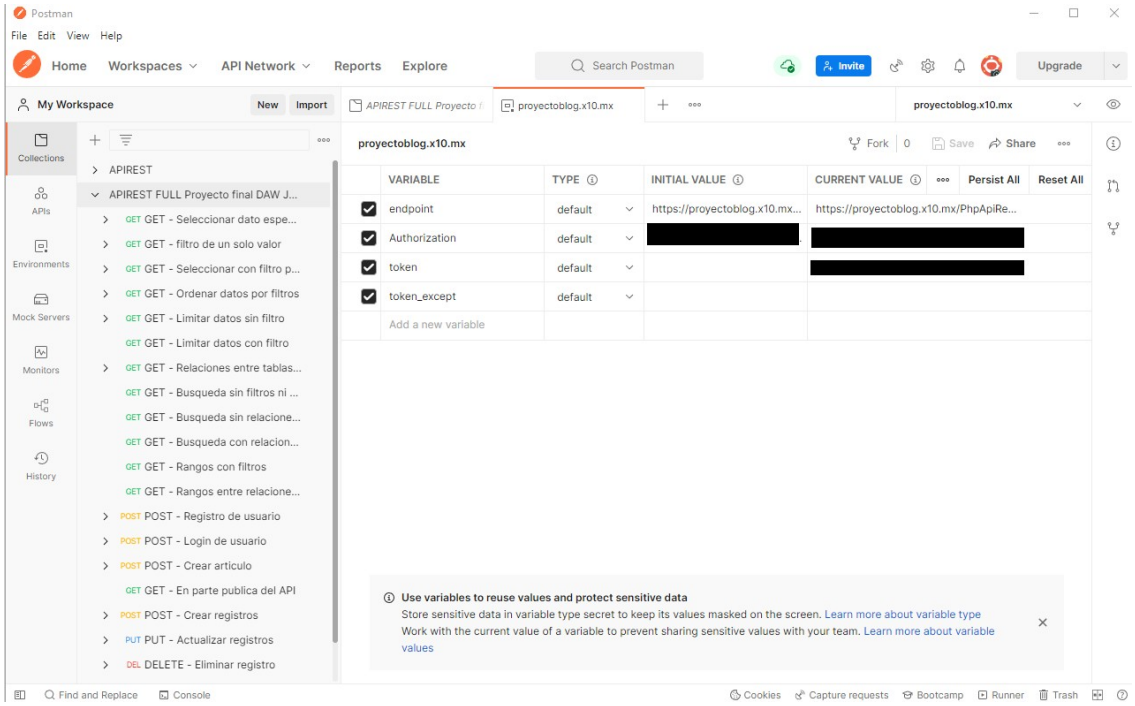


Imagen 6.2.2.11: Ejemplo de como se vería un entorno de Postman configurado

6.2.3 Instalación en un servidor web para producción

6.2.3.1 Como poner en producción la base de datos

Para ello importamos la base de datos tal y como indica el manual de instalación en el apartado 6.2.1.3, y hacemos las configuraciones recomendadas en el apartado 6.2.2.

6.2.3.2 Como poner en producción la API RESTFUL en un servidor Apache con PHP

Una vez configurado debidamente todo según indica el apartado 6.2.2, bastará con copiarlo directamente a la carpeta pública del servidor.

6.2.3.3 Como poner en producción Angular para un servidor Apache

Lo primero es hacer un build del proyecto hecho en Angular (compilación), para ello en una consola ponemos el siguiente comando:

```
ng build --prod --base-href="./"
```

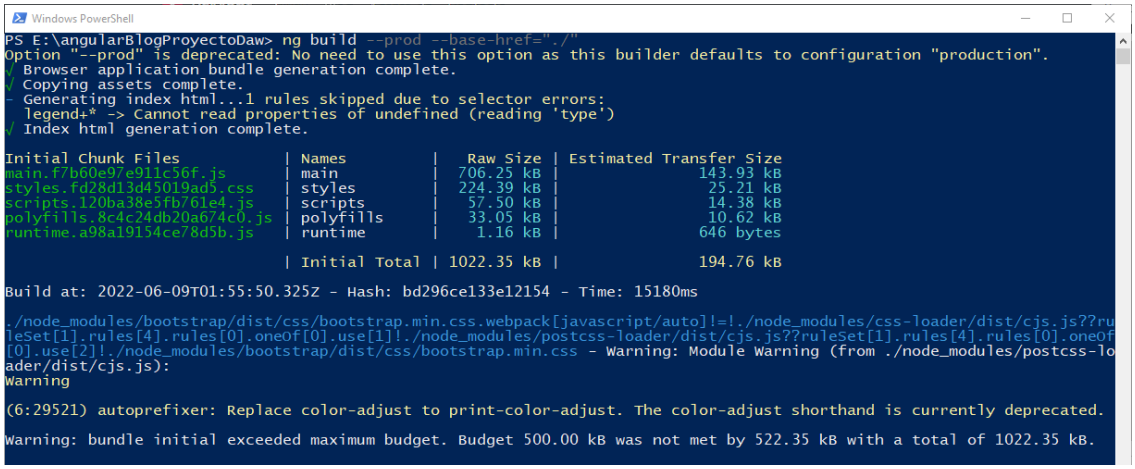


Imagen 6.2.3.3.1: Haciendo una compilación de un proyecto con Angular

Por defecto dentro de la carpeta de proyecto se crea la carpeta **dist** y dentro de ella una carpeta con el nombre del proyecto, entonces si copias el contenido en la raíz de la carpeta pública de un servidor apache debería de funcionar, a la hora de subirlo a un hosting para que sea más inmediato y no de problemas y nos lo permite es recomendable subirlo en un archivo descomprimido y luego descomprimir desde el hosting.

En el caso de que se vaya a meter dentro de una carpeta (algo muy habitual) para que pueda ser ejecutado hay que poner en la carpeta un fichero de configuración de Apache, un fichero llamado ".htaccess" con la siguiente configuración:

```
<IfModule mod_rewrite.c>
  Options Indexes FollowSymLinks
  RewriteEngine On
  RewriteBase /angular-blog-proyecto-daw/
  RewriteRule ^index\.html$ - [L]
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule . index.html [L]
</IfModule>
```

En la linea RewriteBase, hay que indicar la ruta donde situás la compilación en el servidor web, empezando desde la raíz que es la carpeta pública del servidor.

6.3 Manual de usuario

6.3.1 Como Iniciar sesión (Imprescindible para administrar el blog)

Para poder administrar los artículos del blog debes de iniciar sesión con tus credenciales de usuario desde la ruta **login123** por defecto en caso de no haber sido modificada previamente.

TU_DOMINIO/login123

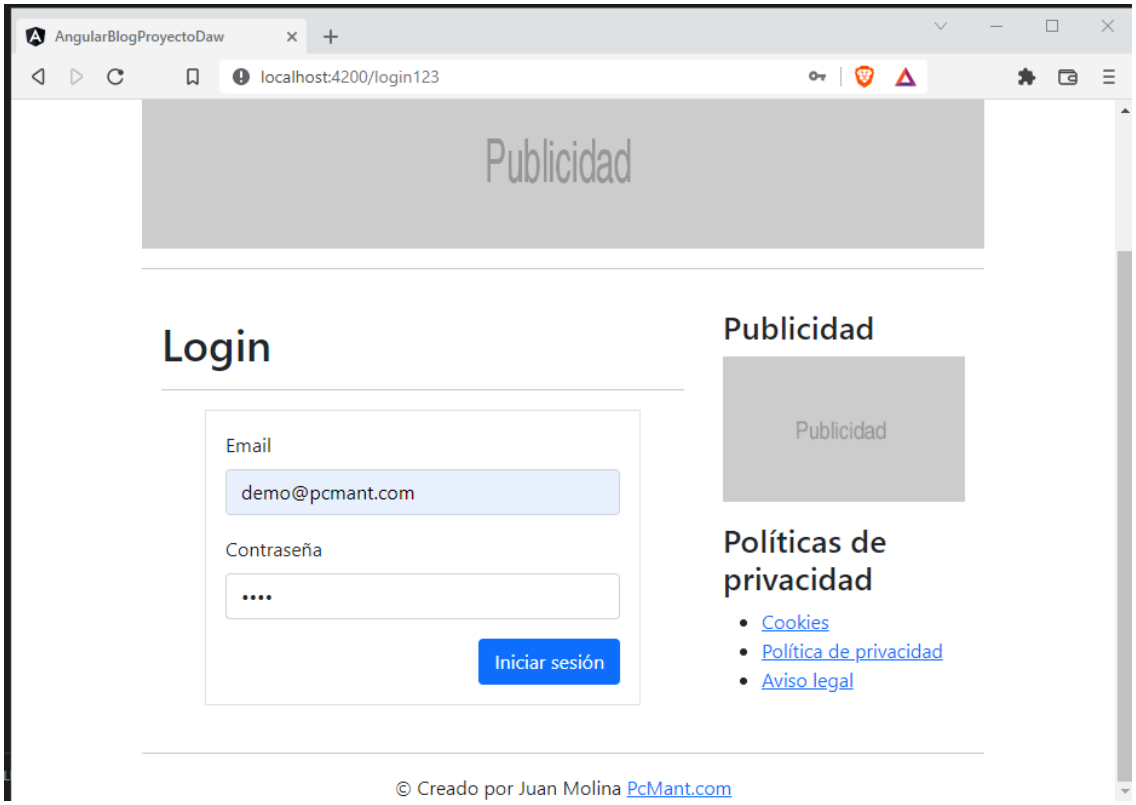


Imagen 6.3.1.1: Iniciando sesión en la aplicación Web creada con Angular

Su sesión durará 24 horas, una vez superadas deberá repetir el procedimiento de inicio de sesión nuevamente.

6.3.2 Como crear un artículo nuevo (Inicio de sesión requerido)

Para crear un artículo nuevo busque el botón verde que hay a un lateral en el cual pone **“Crear artículo”.**

Imagen 6.3.2.1: Crear artículo nuevo desde la aplicación web hecha en Angular

Rellene el formulario con el contenido y dale clic izquierdo al botón guardar, una vez creado le insurreccionará al blog.

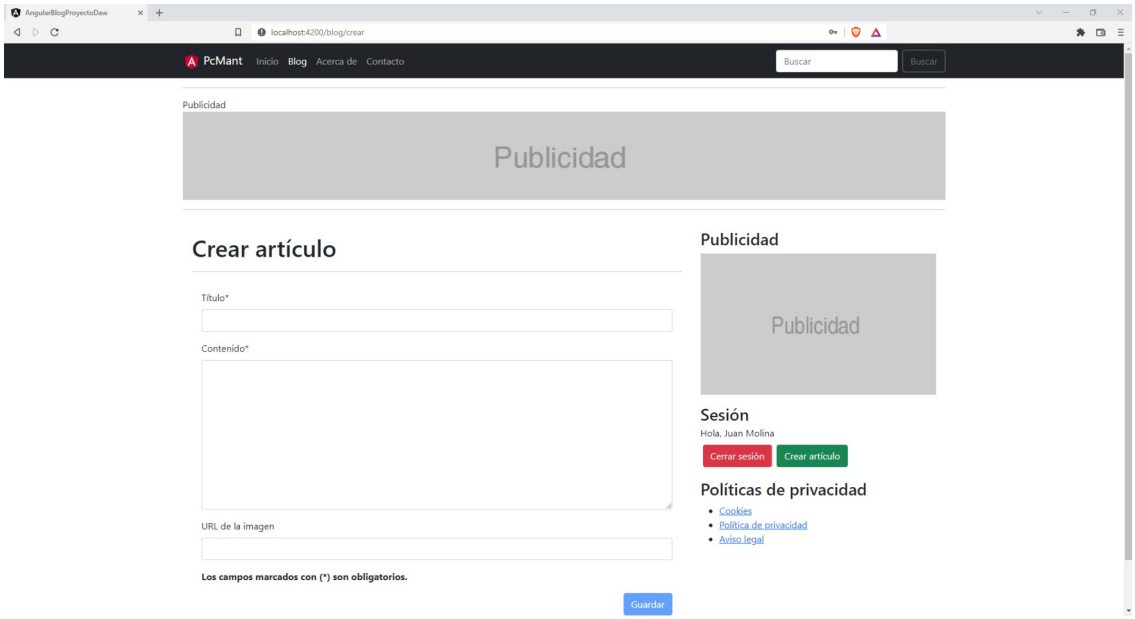


Imagen 6.3.2.2: Formulario de creación de artículos en la aplicación web hecha con Angular

6.3.3 Como Editar/Actualizar un artículo (Inicio de sesión requerido)

Para editar un artículo tiene que hacer clic sobre el y seguidamente al botón **"Editar artículo"**.

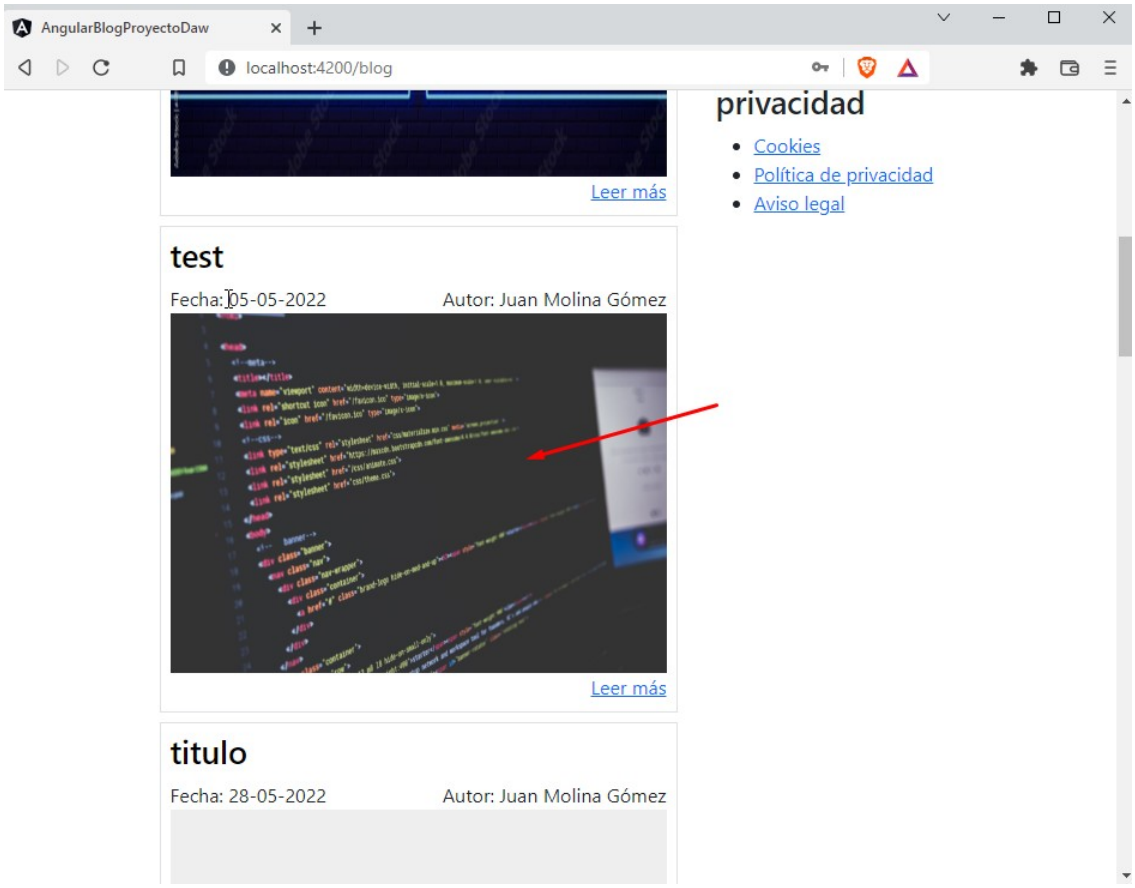


Imagen 6.3.3.1: Como acceder a un artículo desde la aplicación web hecha en Angular

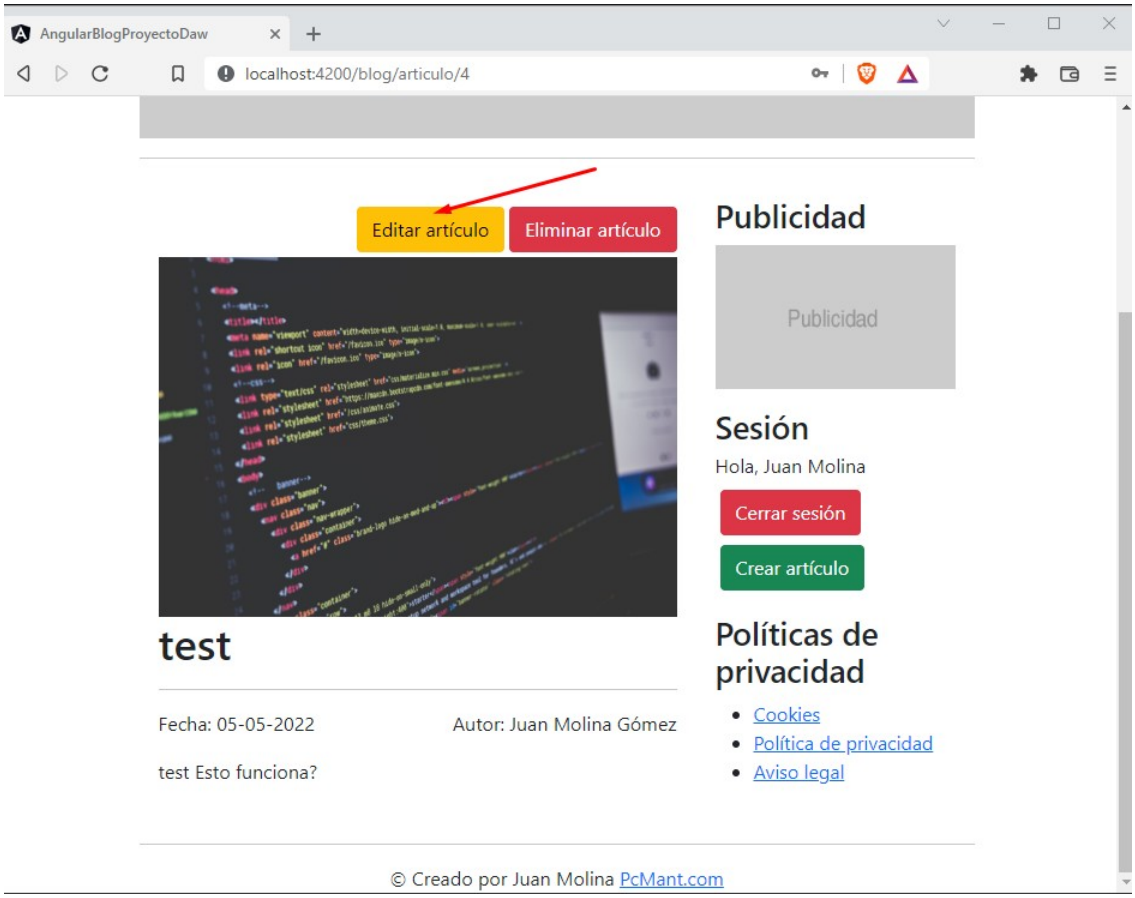


Imagen 6.3.3.2: Como editar un artículo desde la aplicación web hecha con Angular

Modificamos la información del formulario y hacemos clic en guardar.

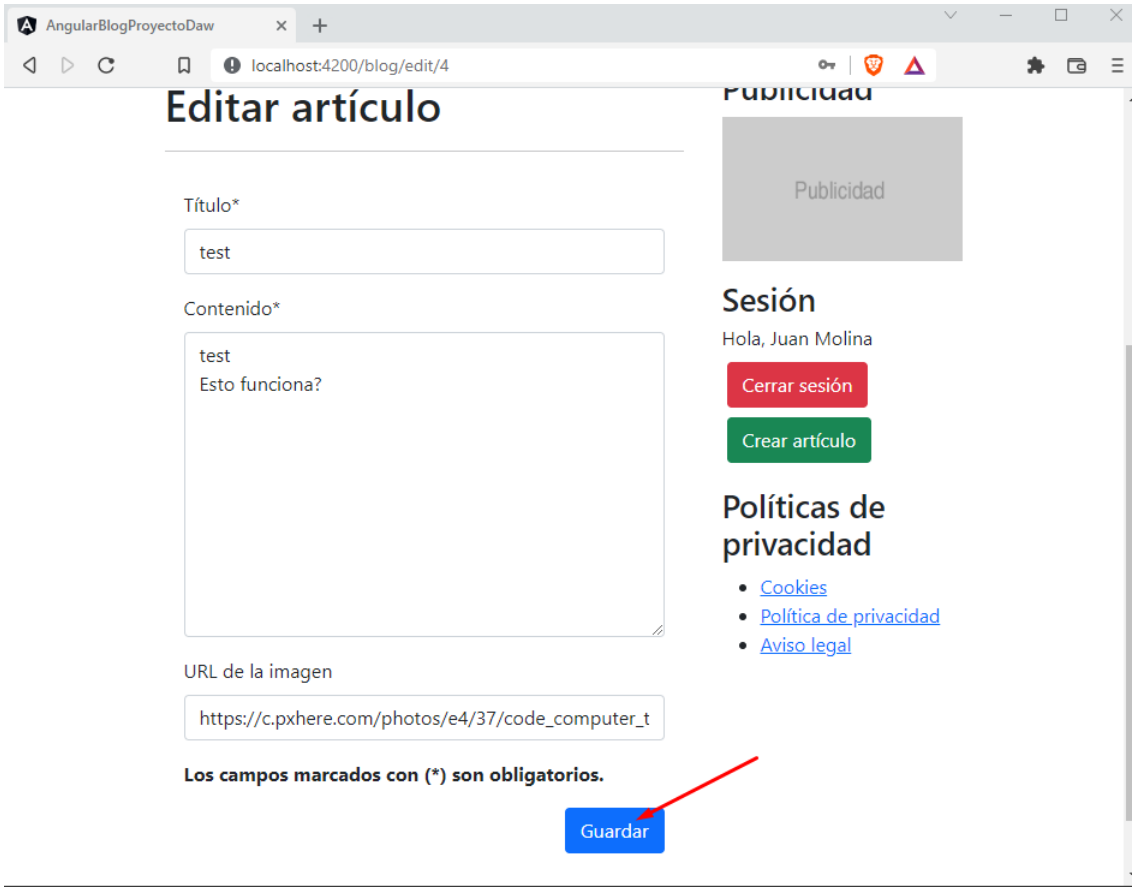


Imagen 6.3.3.3: Formulario de edición de artículo en la aplicación web hecha con Angular

6.3.4 Como eliminar un artículo (Inicio de sesión requerido)

Para borrar un artículo hay que acceder a un artículo en concreto con la sesión iniciada y hacer clic en el botón rojo que pone **"Eliminar artículo"**.

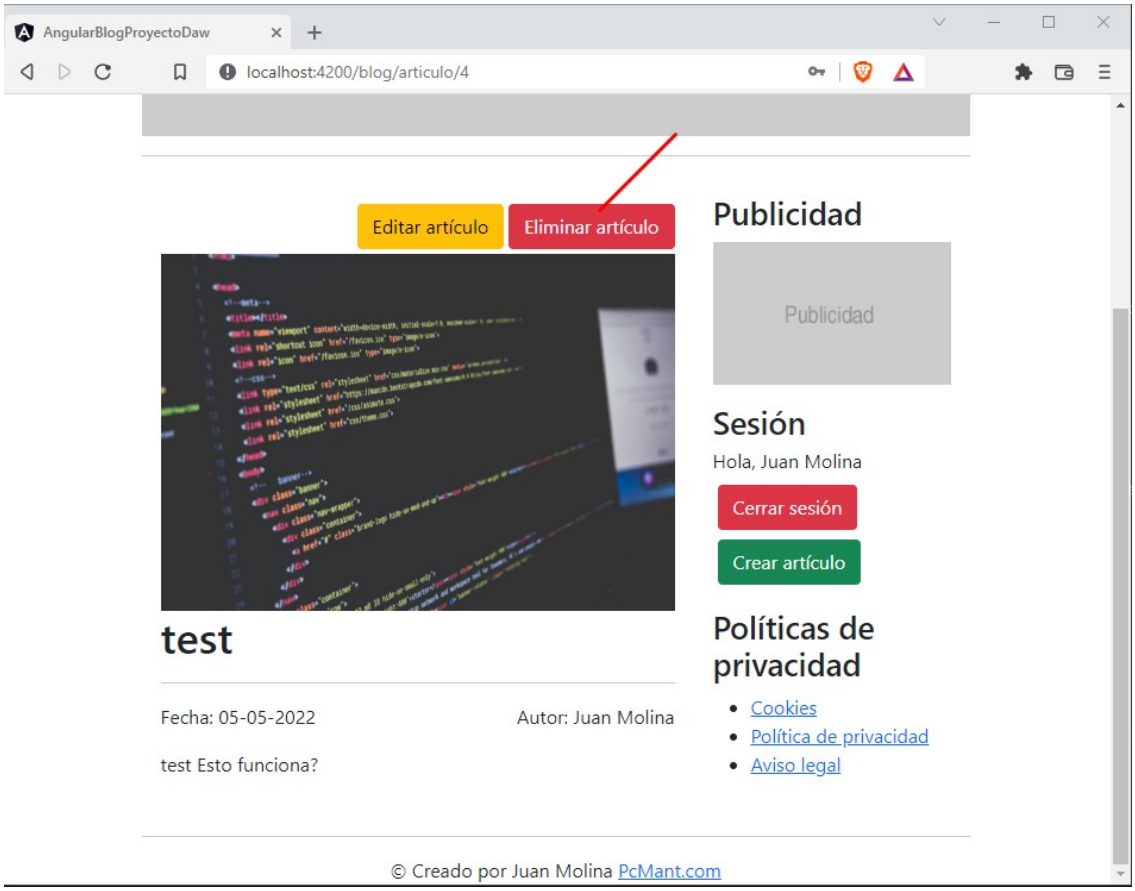


Imagen 6.3.4.1: Como eliminar un artículo en la aplicación web hecha con Angular

6.3.5 Como cerrar la sesión

Por último, si se quiere cerrar la sesión, basta con un clic en el botón rojo que pone **"Cerrar sesión"** y automáticamente serás redireccionado a la home (al inicio).

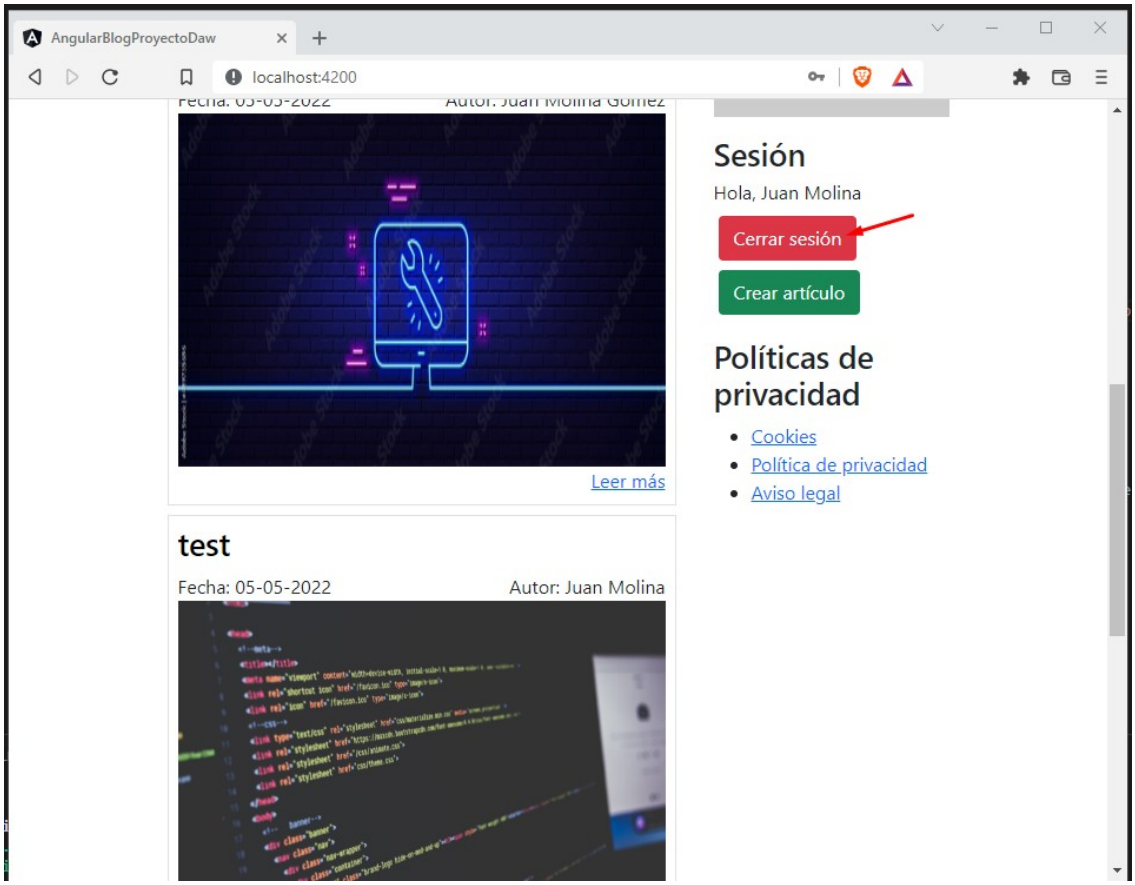


Imagen 6.3.5.1: Como cerrar sesión en la aplicación web hecha con Angular

6.3.6 Uso de Postman

Postman es un cliente HTTP, esto quiere decir que las mismas peticiones que hace Angular a la API las puede hacer dicho cliente, pero a diferencia de este, Postman tiene documentado todo tipo de peticiones con sus parámetros de las solicitudes que le puedes hacer a la API, siempre y cuando hayas importado el fichero JSON con la colección de datos tal y como se indica en el apartado 6.2.2.

Para usarlas tienes que ciclar sobre ellas, meter los valores adecuados a los parámetros según lo que se quiera consultar o datos a modificar (depende del tipo de petición) desde el API.

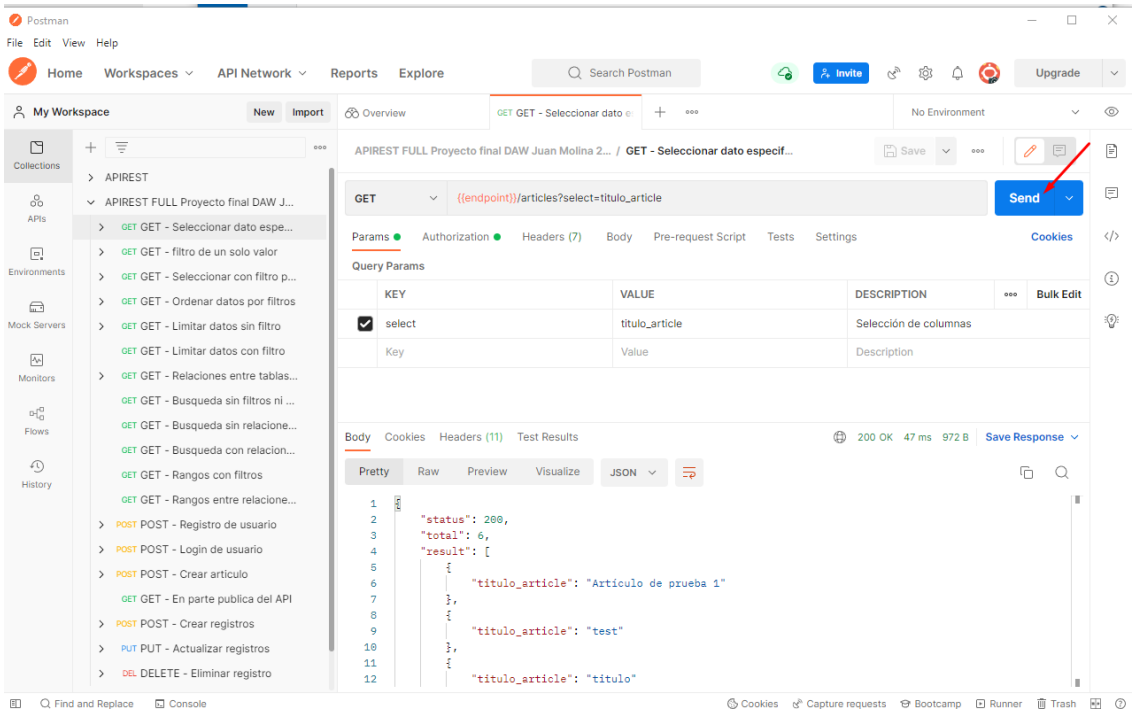


Imagen 6.3.6.1: Haciendo una petición GET a la API desde Postman

Hay funcionalidades que aun no han sido incorporadas al frontend (interfaz gráfica), como es el caso del registro de usuarios o el borrado del mismo por ejemplo, otro requisito para determinadas operaciones es ser conocedor de la base de datos y tener a la mano la documentación que te aporta el fichero de Postman.

Por último y no menos importante, es de vital importancia **antes de hacer cualquier petición POST, PUT o DELETE ejecutar primero el método "POST – Login de usuario" con tus credenciales para obtener la autorización, de lo contrario la petición os dará una respuesta de error el cual indica que no estas autorizado**.

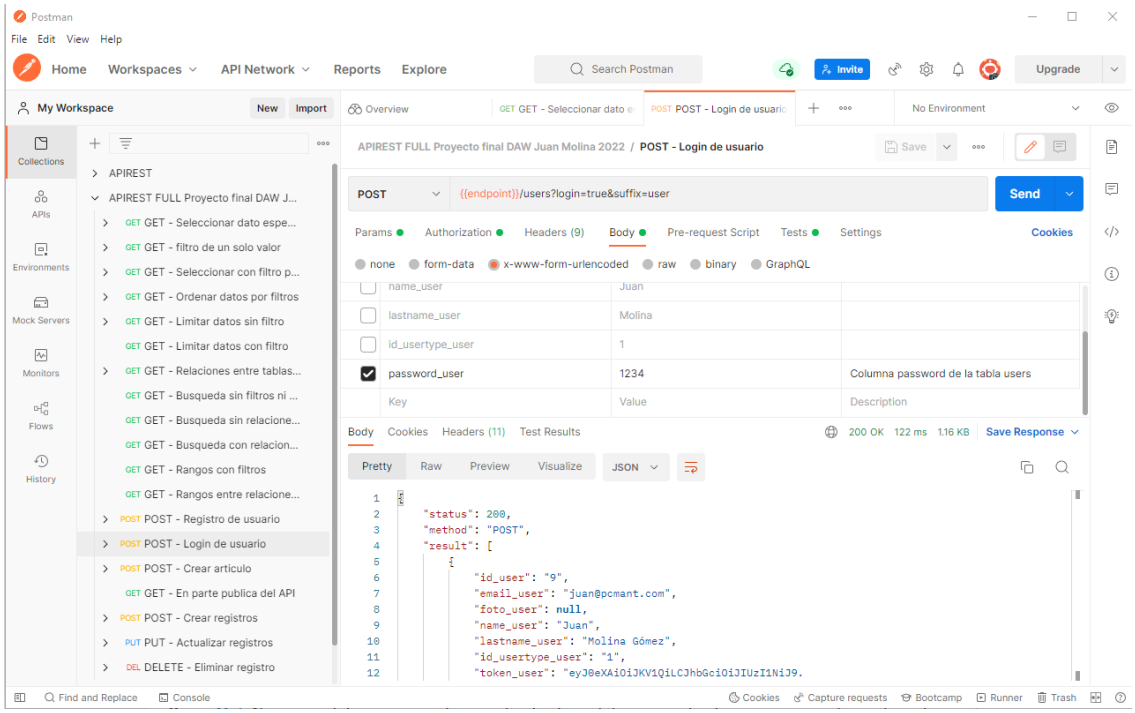


Imagen 6.3.6.2: Iniciando sesión desde Postman

Para aprender más acerca de Postman o consultar cosas acerca de su uso puede ir a la documentación oficial de Postman en https://learning.postman.com/.

7. Conclusiones Finales:

7.1 Evaluación del proyecto: objetivos conseguidos y no alcanzados

La funcionalidad principal se ha conseguido satisfactoriamente, lo que viene siendo el login desde Angular y la gestión de los artículos con un CRUD, aunque me gustaría haber conseguido que fuera más completa la aplicación que consume la API exprimiendo esa API RESTFUL dinámica y la multitud de cosas que permite Angular, pero por factor tiempo y las circunstancias no ha sido posible y me he tenido que conformar, que no es poco.

El tema de las APIS REST con PHP es nuevo para mí, aunque PHP sea una tecnología que ya conocía y Angular en parte, además que es la primera vez que consumo una API RESTFUL propia desde Angular, el cual use PHP y MySQL, lo he notado menos flexible que cualquier otra API hecha en Node.js usando MongoDB o Firebase, pero

como son tecnologías que no manejo, he optado por usar tecnologías que conozco, pero a su vez aprendiendo cosas nuevas.

7.2 Propuestas de mejora

- Editar categorías y seleccionar las mismas.
- Posibilidad de configurar temas de SEO.
- Subida de las imágenes a la propia API.
- Aprovechar la funcionalidad de módulos (Un módulo en Angular es cada una de las aplicaciones que contiene un proyecto e incluso puede compartir servicios) para crear un panel de administración el cual permita configurar el blog.
- Tener un control de usuarios el cual cada uno edite lo suyo y los admins lo de todo y que estos admins puedan gestionar los usuarios.
- Posibilidad de poder poner comentarios. (Tengo en mente que el servicio de terceros llamado Disqus, creo que podría ser una buena opción).
- Tener la opción de que un artículo creado aun no sea publico.
- Al final de cada artículo mostrar foto, nombre y algo descriptivo del autor de dicho artículo.
- Botones en los artículos para compartir.

8. Biografía

- https://angular.io/docs
- https://www.php.net/docs.php
- https://learning.postman.com/
- https://nodejs.org/es/docs/
- https://developer.mozilla.org/
- https://victorroblesweb.es/
- https://www.itsolutionstuff.com/
- https://desarrolloweb.com/
- https://tutorialesatualcance.com/
- http://www.foc.es/
- https://fp.uoc.fje.edu/blog/
- https://openwebinars.net/
- https://jwt.io/
- https://andygeek.com/
- https://dev.to/jumamiller/
- https://programmerclick.com
- https://blog.angular-university.io/
- https://lifesaver.codes/
- https://docs.npmjs.com/3

- https://www.learmoreseekmore.com/
- https://parzibyte.me/blog
- https://si.ua.es/es/documentacion/
- https://www.edix.com/
- https://codigonaranja.com/